TECHNISCHE UNIVERSITÄT
ILMENAU

Fakultät für Informatik und Automatisierung
Institut für Technische Informatik und Ingenieurinformatik
Fachgebiet Softwaresysteme/Prozessinformatik

# Bachelorarbeit

zur Erlangung des akademischen Grades Bachelor of Science

## Klassifikation von Architekturstilen und -mustern hinsichtlich qualitativer Ziele für den Softwarearchitekturentwurf

Autor:
Ralf Stollberg

Betreuer:
Dipl.-Inf. Stephan Bode

Verantwortlicher Hochschullehrer:
Priv.-Doz. Dr.-Ing. habil. Matthias Riebisch

eingereicht am 22.02.2010

## Zusammenfassung

In den vergangenen Jahren haben Architekturstile und -muster als Grundlage für den Softwareentwurf eine große Verbreitung gefunden. Die Auswahl der zu verwendenden Muster während des Entwurfsprozesses basiert bis heute größtenteils auf der Erfahrung des Architekten mit der Anwendung dieser Muster. Bewertungen solcher Architekturlösungsansätze nach ihrem Einfluss auf qualitative Anforderungen, um sie vergleichen zu können, sind rar.

In dieser Arbeit wird eine Hierarchie von Qualitäten, Prinzipien und Attributen, die sich auf die Evolvability auswirken, vorgestellt. Die Beziehungen zwischen den Elementen der Hierarchie wurden gewichtet. Mittels einer Bewertung der Muster nach den unteren Hierarchieebenen kann somit ein Einfluss auf die oberen Hierarchieebenen und schließlich auf die schwer fassbare Evolvability berechnet werden.

Mit dem vorgestellten Bewertungsmodell ist es möglich, Architekturlösungsansätze nach Qualitäten zu beurteilen und untereinander zu vergleichen. In der Arbeit ist dies für eine repräsentative Auswahl bekannter Architekturstile und -muster durchgeführt worden, welche anschließend klassifiziert wurden. Mit den Ergebnissen steht Softwarearchitekten eine hilfreiche Grundlage für die Auswahl der richtigen Muster beim Entwurf einer neuen Software zur Verfügung.

## Abstract

In the last years, usage of architectural styles and patterns have become popular in software design. Architects have to rely on their own experience with those patterns to find their final decision which pattern to use in a certain design. A rating of the influence of such architectural approaches on quality characteristics for comparison is hard to find.

In this bachelor thesis a hierarchy of qualities, principles and attributes, which all affect evolvability, is provided. All influences are weighted, so high-level ratings in the hierarchy can be calculated from the ratings of the lower-level's criteria, which are much more intuitive to evaluate. Finally, a reliable rating of the hardly tangible quality evolvability can be determined.

With the provided hierarchy it is possible to judge and compare architectural approaches by their impacts on qualities. This has been done on a representative selection of established architectural styles and patterns, wich have been classified afterwards. With these results, software architects have a helpful basis for their decision which patterns to use for the design of a new software.

# Inhaltsverzeichnis

1 Einleitung....1
1.1 Motivation....1
1.2 Aufgabenstellung....2
1.3 Definition Evolvability....2
1.4 Abgrenzung....3
1.5 Vorgehen und Ziel dieser Arbeit....3
1.6 Vereinbarungen....3
1.7 Aufbau der Arbeit....4

2 Vorstellung der ausgewählten Muster....5
2.1 Architekturstile....5
Andere Stile....9
2.2 Architekturmuster....10
Benutzerschnittstelle....10
Ablaufsteuerung....11
Schnittstellen....13
Verteilung....14
Adaptierbarkeit....14

3 Vorgehen bei der Bewertung....17
3.1 Zerlegung von Evolvability....17
3.1.1 Definitionen....17
Prinzipien und Attribute....17
Qualitäten....18
3.1.2 Hierarchie der Bewertungskriterien....19
Erläuterung der Einflüsse....21
3.2 Bewertungsskala....24
3.3 Bewertbarkeit der Kriterien....24
Prinzipien und Attribute....24
Qualitäten....27

4 Bewertung der Muster....28
4.1 Die Musterbeispiele....28

4.1.1 Beispiel 1: Verwaltungsprogramm für Sammelbesteller....28
4.1.2 Beispiel 2: Aufnahme eines Videos....29
4.2 Architekturstile....29
Client-Server....29
Layers, Tiers....31
Repository....33
Blackboard....35
Batchprogramme....36
Pipes and Filters....38
4.3 Architekturmuster....40
4.3.1 Benutzerschnittstelle....40
Model-View-Controller (kurz MVC)....40
Presentation-Abstraction-Control (kurz PAC)....42
4.3.2 Ablaufsteuerung....43
Event-Based, Implicit Invocation....43
Nebenläufigkeit....44
Inversion of Control....45
4.3.3 Schnittstellen....47
Fassade....47
Adapter....48
Dependency Injection....50
4.3.4 Verteilung....51
Broker....51
Proxies....52
4.3.5 Adaptierbarkeit....54
Mikrokernel....54
Reflection....56
Plug-In....58

5 Auswertung und Klassifikation....60
5.1 Auswertung der Bewertungen....60
5.1.1 Beschreibung der Berechnungsvorlage....60
Die Tabelle "Abhängigkeiten"....61
Die Tabelle "Ergebnisse"....62
Endergebnis....63
5.1.2 Interpretation der Bewertungen....65
Geringer Betrag der Bewertung....65

Tauglichkeit der Muster für die Evolvability....66
5.1.3 Nutzen der Bewertungen....69
5.2 Klassifikation....70
5.2.1 Entstehung der Klassen....70

6 Fazit und Ausblick....72

7 Anhang....74
7.1 Ergebnisse der Bewertung....74
7.2 Darstellung der Klassen....75
7.3 Literaturverzeichnis....79
7.4 Selbständigkeitserklärung....83

# 1 Einleitung

## *1.1 Motivation*

In den vergangenen Jahren hat sich eine breite Community gebildet, die diverse Architekturstile und -muster zusammengetragen hat. In vielen Jahren sind seither immer wieder neue Muster aufgetaucht und sogar ganze Mustersysteme erstellt worden (z.B. [BMRSS96]). Die Auswahl der zu verwendenden Muster während des Entwurfsprozesses basiert bis heute größtenteils auf der Erfahrung des Architekten mit ihrer Anwendung. Bewertungen nach ihrem Einfluss auf qualitative Anforderungen, um sie vergleichen zu können, sind rar. Ziel dieser Arbeit soll es sein, Architekturstile und -muster auf ihre Einflüsse auf die Evolvability zu bewerten. Da Evolvability ein sehr umfassender Begriff ist, wird sie dazu in leichter zu fassende Qualitäten zerlegt. Untersucht wird eine repräsentative Auswahl der wichtigsten Stile und Muster, die sich auf die Evolvability auswirken.

Die Vorzüge einer solchen Übersicht liegen auf der Hand: Der Softwarearchitekt erhält dadurch Unterstützung bei der Auswahl der Muster. Natürlich soll und kann diese Bewertung nicht die alleinige Grundlage von Entscheidungen sein, viel zu groß ist die Menge der zu beachtenden Einflussfaktoren und Wechselwirkungen zwischen einzelnen Mustern.

Der Autor beschäftigt sich erstmals intensiv mit dem Thema Softwarearchitektur und verschafft sich mit dieser Arbeit einen tiefgreifenden Einblick in die behandelten Muster und ihre Auswirkungen. Er hat schon an der eigenen Programmiertätigkeit erfahren, dass ein Softwareentwurf ohne dieses Wissen wesentlich schwieriger ist und die Qualität des Resultats entscheidend davon geprägt wird.

## *1.2 Aufgabenstellung*

Klassifikation von Architekturstilen und -mustern hinsichtlich qualitativer Ziele für den Softwarearchitekturentwurf

Entwurfsprinzipien sind ein wichtiges Hilfsmittel, von denen sich ein Softwarearchitekt beim Architekturentwurf leiten lässt. Weiterhin nutzt der Architekt bekannte Architekturstile und -muster oder Frameworks, die als Lösungen zu verschiedenen Prinzipien und Architekturzielen beitragen. Für den Softwarearchitekturentwurf ist es wichtig, speziell für qualitative Ziele und Anforderungen geeignete Prinzipien und Lösungen zu finden, um diese Ziele in der Architektur adäquat umzusetzen. Ziel dieser Arbeit ist die Klassifikation von Architekturstilen und -mustern nach ihrem Beitrag zur Umsetzung von vorgegebenen Architekturprinzipien und qualitativen Zielen. Dazu sind typische Architekturstile und -muster entsprechend ihrem Beitrag zur Umsetzung von Entwurfsprinzipien und qualitativen Zielen zu bewerten und zuzuordnen.

Diese Arbeit soll sich mit denjenigen Entwurfsprinzipien und qualitativen Zielen beschäftigen, die sich auf die Evolvability der Software positiv oder negativ auswirken. Welche dies im Einzelnen sind, wird im Kapitel 3 - Vorgehen bei der Bewertung genauer beschrieben.

## *1.3 Definition Evolvability*

"Software evolvability is the ability of a software system to adjust to change stimuli, i.e. changes in requirements and technologies that may have impact on the software system in terms of software structural and/or functional enhancements, while still taking the architectural integrity into consideration." [BCE07]

Eine Software hat also genau dann eine gute Evolvability, wenn sie ohne großen Aufwand an veränderte funktionelle oder strukturelle Anforderungen oder Technologien angepasst werden kann, ohne dabei ihre architekturelle Integrität zu verlieren.

## 1.4 Abgrenzung

Diese Arbeit soll und kann nicht die Gesamtheit aller relevanten Architekturstile und -muster betrachten. Dies liegt zum einen an der unscharfen Abgrenzung und zum anderen an der sich ständig ändernden Menge der Stile und Muster. Der Autor ist aber bemüht, eine repräsentative Auswahl zu treffen. Der Schwerpunkt der Arbeit liegt nicht auf einer umfassenden Katalogisierung vorhandener Stile und Muster, sondern auf der Bewertung der getroffenen Auswahl und der Vorstellung eines Bewertungsmechanismusses.

## 1.5 Vorgehen und Ziel dieser Arbeit

Es werden Architekturprinzipien, Attribute und qualitative Ziele ausgewählt, die sich unterstützend oder hemmend auf die Evolvability von Software auswirken. Aus ihnen wird eine Hierarchie gebildet. Die Beziehungen zwischen den Elementen der Hierarchie werden gewichtet, so dass aus einer Bewertung der unteren Level der Einfluss auf die Evolvability errechnet werden kann. Anschließend wird eine repräsentative Auswahl von Architekturstilen und -mustern nach diesen Prinzipien, Attributen und Qualitäten bewertet, um Rückschlüsse auf ihre Auswirkung auf die Evolvability der daraus resultierenden Software zu ziehen. Diese Bewertung wird anhand von beispielhaften Architekturentwürfen erläutert, um eine möglichst hohe Verständlichkeit zu erreichen. Anhand dieser Bewertungen und der Hierarchie wird mittels dem unter dem Namen Berechnungsvorlage.ods auf CD beigelegten Tabellendokument das Ergebnis bezüglich Evolvability ermittelt, welches als Entscheidungshilfe beim Entwurf einer Softwarearchitektur verwendet werden kann. Zudem wird eine Klassifizierung der Muster vorgestellt.

## 1.6 Vereinbarungen

Wenn der Autor in dieser Arbeit allgemein von Mustern spricht, so sind Architekturstile und Architekturmuster gleichermaßen gemeint.

Die Evolvability wird in Abschnitt 3.1 - Zerlegung von Evolvability in Subcharakteristiken zerlegt. Wird allgemein von Bewertungskriterium gesprochen, so ist eine dieser Qualitäten, Prinzipien oder Attribute gemeint, ohne sie näher zu spezifizieren.

## *1.7 Aufbau der Arbeit*

Das einleitende Kapitel enthält Motivation, Aufgabenstellung und die für diese Arbeit wichtige Definition der Evolvability sowie eine Abgrenzung, das Vorgehen und die Ziele dieser Arbeit und begriffliche Vereinbarungen.

Das zweite Kapitel wird die später zu bewertenden Muster vorstellen. Anschließend werden im dritten Kapitel die Bewertungskriterien definiert und eine Hierarchie daraus gebildet, anhand der die Abhängigkeiten unter ihnen dargestellt und später auch verrechnet werden. Da die Hierarchie dafür ausgelegt ist, alle Einflüsse auf die resultierende Software zu bewerten, wird am Ende des dritten Kapitels die Anwendbarkeit der einzelnen Kriterien auf Muster diskutiert.

Nachdem im vierten Kapitel die Bewertung stattgefunden hat, werden am Anfang des fünften Kapitels die Funktionsweise der Berechnungsvorlage erläutert, die Ergebnisse präsentiert und die Muster klassifiziert.

Zum Abschluss wird ein Fazit gezogen und ein Ausblick auf mögliche weitergehende Arbeiten gegeben.

# 2 Vorstellung der ausgewählten Muster

## *2.1 Architekturstile*

### Client-Server

Es existiert ein Serverprozess, der Dienste zur Verfügung stellt. Einer oder mehrere Clientprozesse basieren auf den Diensten des Servers. [wikiCS] Siehe auch Abbildung 1: Client-Server

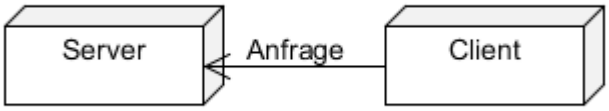


*Abbildung 1: Client-Server*

Das Client-Server Modell hat sich seit vielen Jahrzehnten in Systemen etabliert. Es ist heutzutage ein Grundbaustein, der in vielen Stilen und Mustern Verwendung findet. Dieser Stil ist sowohl auf Prozesse anwendbar, die auf einem Rechner laufen, als auch auf Prozesse, die in einem Netzwerk verteilt sind.

### Layers, Tiers

Jede Schicht (engl. Layer bzw. Tier) stellt Dienste für die nächsthöhere Schicht bereit und nutzt Dienste der darunterliegenden Schicht. Verbindungen zwischen nicht benachbarten Schichten sind meist verboten. [SG96] Layers ist eine logische Unterteilung (alle Schichten laufen auf dem gleichen Rechner), während Tiers eine physikalische Unterteilung darstellt (Verteilung im Netzwerk). [msdn998478] Siehe auch Abbildung 2: Layers und Abbildung 3: Tiers

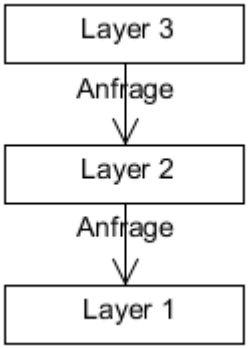


*Abbildung 2: Layers*

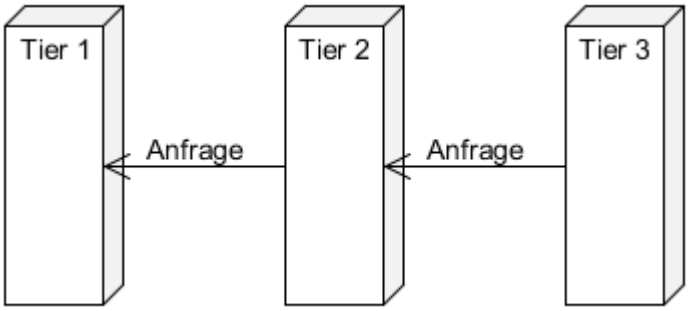


*Abbildung 3: Tiers*

Layers besteht aus einer Hierarchie von Client-Server-Beziehungen.

**Repository**

Mehrere unabhängige Prozesse haben Zugriff auf eine zentrale Datenbasis (das Repository). Das Repository verhält sich dabei passiv, die Prozesse werden durch entsprechende Aktionen anderer Prozesse aktiviert. [SG96] Siehe auch Abbildung 4: Repository

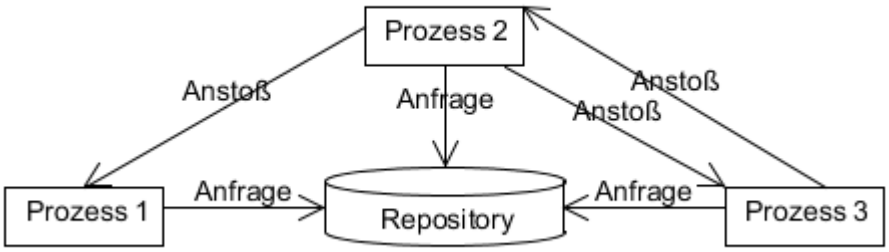


*Abbildung 4: Repository*

Tatsächlich kann schon eine zentrale Datei, auf der mehrere Prozesse arbeiten, als Repository angesehen werden. Ebenso kann es sich um eine Datenbank handeln. Das Repository ist hierbei der Server für Datendienste, die darauf arbeitenden Prozesse die Clients.

### Blackboard

Mehrere unabhängige Prozesse haben Zugriff auf eine zentrale Datenbasis (das Blackboard). Das Blackboard aktiviert die Prozesse bei entsprechenden Änderungen an Datensätzen, es handelt also aktiv. [SG96] Siehe auch Abbildung 5: Blackboard

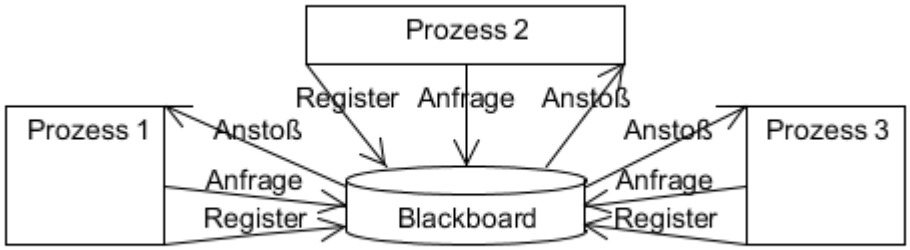


*Abbildung 5: Blackboard*

Im Unterschied zum Repository hat das Blackboard eine zusätzliche Funktionalität, um Registrierungen entgegenzunehmen. Mit der Registrierung werden Bedingungen definiert, wann der sich registrierende Prozess vom Blackboard informiert werden soll. Meist ist die Bedingung eine Änderung an einem bestimmten Datensatz. Genau diese zusätzliche Funktionalität ändert das Verhalten des Blackboards gegenüber dem des Repository dermaßen, dass es hier im Hinblick auf die Evolvability gesondert behandelt wird.

### Batchprogramme

Mehrere externe Programme werden nacheinander ausgeführt. Bevor das darauffolgende Programm startet, muss das aktuelle Programm beendet sein. Somit ist zu jeder Zeit nur ein Programm aktiv. Die Daten (der Batch) werden jeweils als Ganzes von Programm zu Programm weitergereicht. [BCK98] Siehe auch Abbildung 6: Batchprogramm

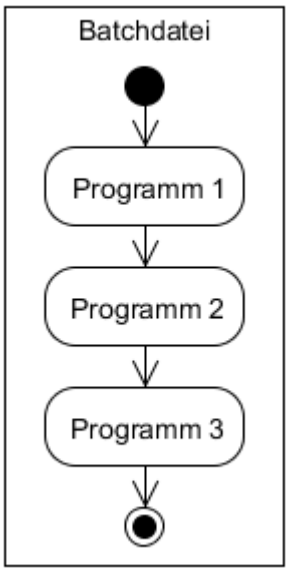


*Abbildung 6:*
*Batchprogramm*

Die Abarbeitung von Batchprogrammen ist ebenso wie das Client-Server Modell eine sehr alte Vorgehensweise. In Betriebssystemen ohne grafische Benutzeroberfläche hat man eine Menge von spezialisierten Dienstprogrammen für elementare Aufgaben zur Verfügung. Mit Batchprogrammen lassen sich oft wiederkehrende Aufgabenketten durch einen einzigen Befehl in der Konsole starten.

### Pipes and Filters

Jeder Filter hat eine Menge von Eingängen und eine Menge von Ausgängen. Er führt (typischerweise inkrementell) Berechnungen auf den Eingangsdaten aus und stellt sie am Ausgang wieder zur Verfügung. Sobald am Ausgang die ersten Daten vorliegen, beginnt der jeweils nachfolgende Filter mit seinen Berechnungen. Somit sind nach einer Anlaufphase typischerweise alle Filter in der Filterkette parallel aktiv. Die Berechnungen der Filter sind unabhängig von denen anderer Filter. Die Verbindungen zwischen den Filtern werden Pipes genannt und sind lediglich für den Datentransport zuständig. [SG96] Siehe auch Abbildung 7: Pipes and Filters

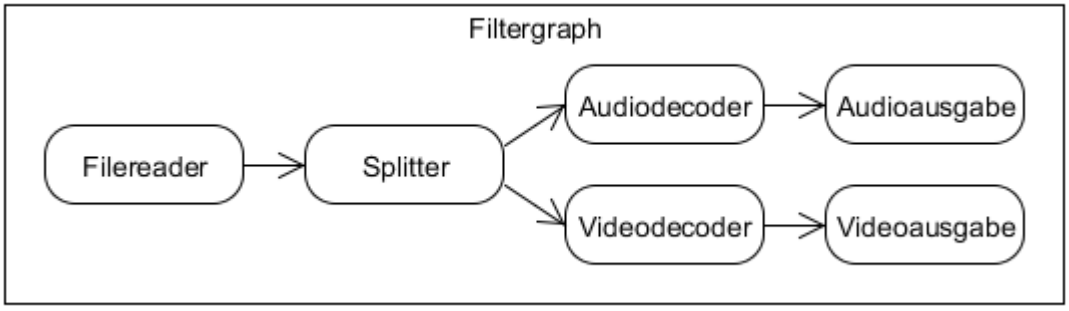


*Abbildung 7: Pipes and Filters*

Pipes and Filters ist eine Weiterentwicklung von Batchprogrammen, die in bestimmten Fällen große Vorteile bietet. Diese treten besonders dann in Erscheinung, wenn die Daten in unabhängige Chunks unterteilt werden können (z.B. Video- oder Audioframes). Je kleiner diese Chunks sind, desto schneller stehen am Ausgang der Filterkette erste Ergebnisse bereit. Die oben genannte Definition beschreibt eine Push-Filterkette. Bei der sogenannten Pull-Filterkette fordert der Ausgabefilter die Daten chunkweise vom vorhergehenden Filter an, der wiederum eine Anforderung an den vorhergehenden Filter sendet. So wird ein Pufferüberlauf vermieden, indem jeder Filter nur so viele Daten produziert, wie sofort vom jeweils nächsten Filter konsumiert werden können. Auch Echtzeitanforderungen sind mit Pull-Filterketten wesentlich einfacher realisierbar, die Bedingungen befinden sich dann als Metadaten in den Chunk-Anforderungen.

Neben der schnellen Verfügbarkeit der ersten Ergebnisse ist ein weiterer Vorteil die parallele Berechnung in mehreren Prozessen (typisch einer pro Filter). Dadurch können die Vorteile moderner Mehrprozessorsysteme genutzt werden.

#### *Andere Stile*

Es gibt noch einige weitere Stile, auf deren Behandlung verzichtet wird, weil sie für das Thema Evolvability nicht relevant sind. Beispielsweise besteht ein Peer-to-Peer Netzwerk aus vielen gleichartigen Clients [wikiP2P]. Kernproblem ist die Verwaltung der Adressen anderer Peers. Dieses Problem ist aber orthogonal zur Evolvability. Für die Evolvability ist lediglich die Architektur der Peers selbst interessant, welche aber nicht Gegenstand der Definition von Peer-to-Peer Netzwerken ist.

## 2.2 Architekturmuster

### *Benutzerschnittstelle*

#### Model-View-Controller

Die Anwendung wird aufgeteilt in Modell und Benutzerschnittstelle. Das Modell enthält die komplette Kernfunktionalität. Die Benutzerschnittstelle ist weiter unterteilt in die Darstellung (View) und die Steuerung (Controller). Die Darstellung stellt die Ausgabeschnittstelle (meist Bildschirmausgabe) bereit. Die Steuerung verarbeitet die Eingaben (meist von Tastatur und Maus), interagiert mit der Darstellung und steuert das Modell. Benachrichtigungsmechanismen stellen die Konsistenz zwischen Benutzerschnittstelle und Modell sicher. [BMRSS96] Siehe auch Abbildung 8: Model-View-Controller

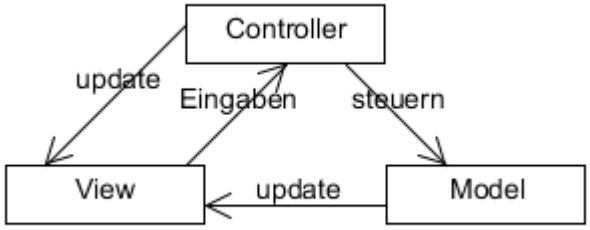


*Abbildung 8: Model-View-Controller*

#### Presentation-Abstraction-Control

Die Anwendung wird in mehrere miteinander kommunizierende (hierarchisch angeordnete) Funktionseinheiten unterteilt. Jede Funktionseinheit wird wiederum unterteilt in Abstraktion (beinhaltet die abstrakte Funktionalität), Präsentation (Benutzerschnittstelle) und Kommunikationsmodul zu den anderen Funktionseinheiten (Control). [BMRSS96] Siehe auch Abbildung 9: Presentation-Abstraction-Controller

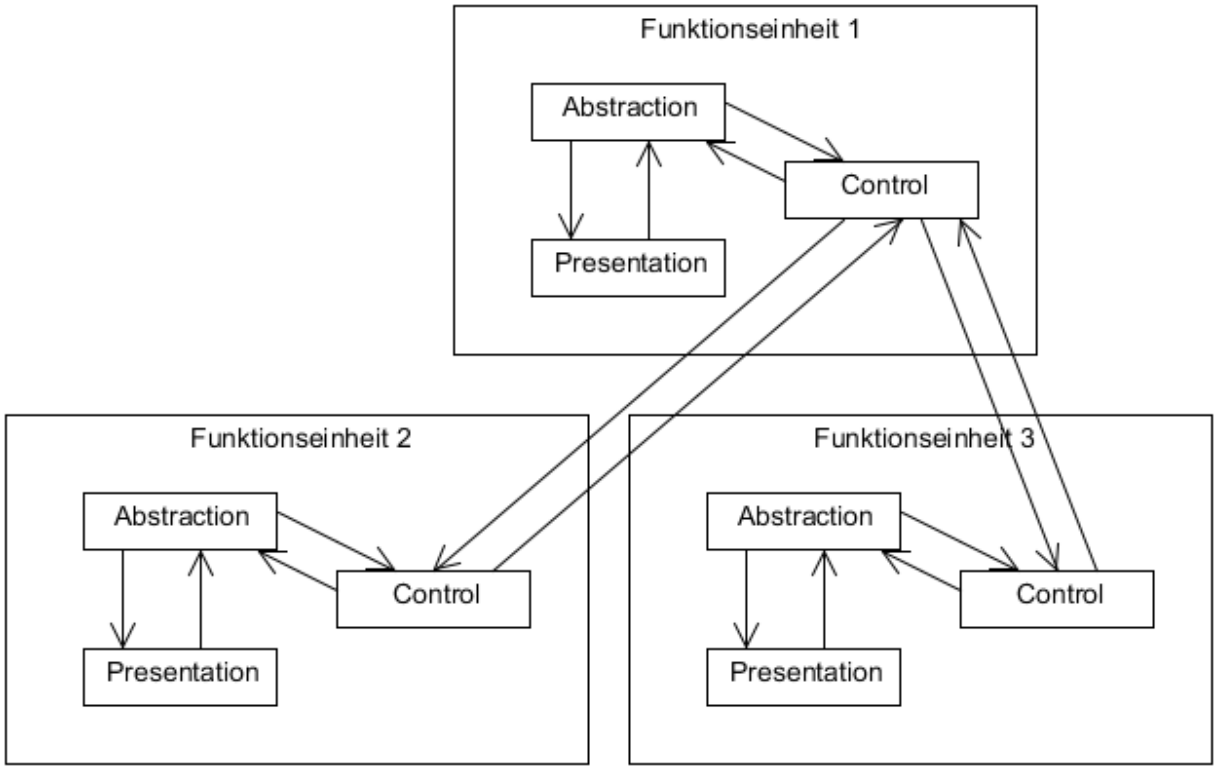


*Abbildung 9: Presentation-Abstraction-Controller*

### *Ablaufsteuerung*

#### Event-Based, Implicit Invocation

Anstatt eine Prozedur direkt aufzurufen, wird das Auftreten eines bestimmten Ereignisses propagiert. Andere Komponenten der Software können sich dafür registrieren, dass bei Auftreten eines Ereignisses eine Prozedur als Ereignisbehandlungsroutine aufgerufen wird. [SG96]

Ereignisse sind ein typisches Merkmal Objektorientierter Programmierung. Sie spielen ebenso eine Schlüsselrolle bei Inversion of Control.

#### Nebenläufigkeit

Moderne Systeme müssen jederzeit auf Eingaben reagieren können, auch wenn sie gerade Berechnungen ausführen. Um das zu erreichen, benötigt man Nebenläufigkeit.Es haben sich mehrere Konzepte, wie beispielsweise Prozesse oder Threads, entwickelt, die

sich dieser Problematik annehmen. Im Hinblick auf die Evolvability ist es allerdings von untergeordneter Bedeutung, welcher dieser Ansätze benutzt wird, um Nebenläufigkeit zu realisieren. Daher wird hier nicht zwischen ihnen unterschieden. Es wird stellvertretend für alle Konzepte die Bezeichnung Prozess benutzt werden.

### Inversion of Control

Die althergebrachte Methode zur Einbindung externer Bibliotheken gleicht einem Prozeduraufruf. Die externe Bibliothek bearbeitet die Aufgabe sofort und gibt die Steuerung danach sofort zurück. Die Ablaufsteuerung bleibt somit im aufrufenden Programm selbst. Frameworks hingegen arbeiten nach Inversion of Control. Sie übernehmen nach der Initialisierungsphase die Ablaufsteuerung des Programms und binden es lediglich noch über (in der Initialisierungsphase registrierte) Ereignisbehandlungsroutinen ein (Event-Based, Implicit Invocation). Die Ablaufsteuerung (engl. Control) ist somit invertiert. [FowIoC] Siehe auch Abbildung 10: Inversion of Control

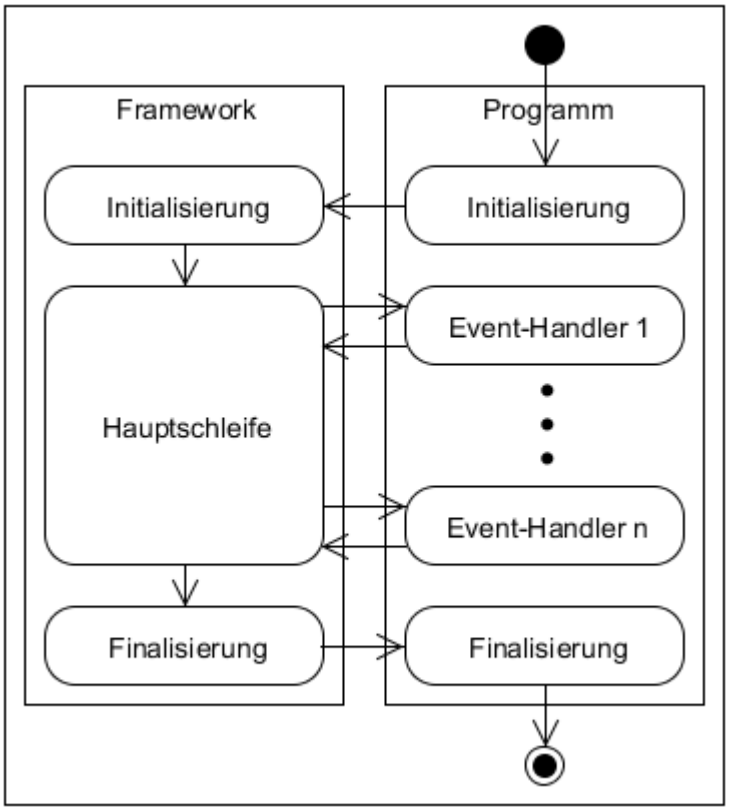


*Abbildung 10: Inversion of Control*

Inversion of Control steht in der Literatur oft in engem Zusammenhang mit Dependency Injection. Nach Ansicht des Autors ist ein gewisser Zusammenhang zwar nicht von der Hand zu weisen, dennoch werden sie losgelöst voneinander in eigenen Abschnitten betrachtet.

In der Literatur ist die Einordnung von Inversion of Control als Architekturprinzip oder Architekturmuster umstritten. Die Einordnung zwischen Architekturmustern soll nicht als Ansicht des Autors zu dieser Frage angesehen werden, es passt an dieser Stelle lediglich gut in die Struktur der Arbeit.

### *Schnittstellen*

#### Fassade

Eine Fassade stellt eine zusammengefasste und vereinfachte Schnittstelle für ein Subsystem mit mehreren einzelnen komplexen Schnittstellen zur Verfügung, ohne die einzelnen Schnittstellen des Subsystems dabei zu verstecken. [GHJV95]

Die Fassade dient vornehmlich der Vereinfachung der Schnittstelle (Verständlichkeit) sowie der Entkopplung.

#### Adapter

Ist eine Clientschnittstelle nicht kompatibel zu einer Serverschnittstelle, so wird ein Adapter zwischengeschaltet, der entsprechende Transformationen vornimmt. Dabei kann auch geringfügig zusätzliche Funktionalität ins Spiel kommen. [GHJV95]

Laut Gamma et al. ist Wrapper lediglich ein Synonym für den Adapter. Nach Meinung des Autors ist ein Wrapper zwar eher ein Adapter, der die Komponente (z.B. durch Vererbung) vollständig kapselt und so einen direkten Zugriff verhindert, allerdings ist dieser feine Unterschied in Bezug auf die Evolvability eher irrelevant, weshalb hier lediglich der Adapter behandelt wird.

#### Dependency Injection

Benötigt eine Komponente eine andere Komponente, um ihre Arbeit zu tun, so nennt man das eine Abhängigkeit (engl. Dependency). Die benutzende Komponente definiert das Interface der zu benutzenden Komponente. Um Funktionen des Interfaces nutzen zu können, muss nun lediglich noch eine konkrete Implementierung des Interfaces von außen

in diese Komponente injiziert (engl. Injection) / ein Zeiger auf eine Instanz eines entsprechenden Objektes übergeben werden. [FowDI]

Dependency Injection steht in der Literatur oft in engem Zusammenhang mit Inversion of Control. Nach Ansicht des Autors ist ein gewisser Zusammenhang zwar nicht von der Hand zu weisen, dennoch werden sie losgelöst voneinander in eigenen Abschnitten betrachtet.

### *Verteilung*

#### Broker

Ein Vermittler (engl. Broker) managt die Kommunikation zu Dienstanbietern auf entfernten Systemen. Die entfernten Systeme melden sich dynamisch beim Vermittler an und propagieren ihre angebotenen Dienste. Diese stellt der Vermittler dynamisch einem lokalen Dienstnutzer zur Verfügung. Der Dienstnutzer ist somit vom Dienstanbieter entkoppelt. [BMRSS96]

#### Proxies

Ein Proxy repliziert Dienste eines Servers und entlastet ihn dadurch. Dabei bleibt das Interface unverändert. Eine Entlastung des Servers wird z.B. durch Zwischenspeicherung (Caching) häufiger Dienstanfragen oder Überprüfung von Sicherheitsrichtlinien und ggf. Zurückweisung der Dienstanfragen erreicht, bevor die Anfrage den Server erreicht. Zudem wird eine Entkopplung von Server und Dienstnutzer erreicht. [BMRSS96]

### *Adaptierbarkeit*

#### Mikrokernel

Der Mikrokernel ist eine Komponente, die nur atomare, grundlegende Dienste eines Systems enthält. Weitergehende Funktionalität und Funktionalität mit Abhängigkeiten zum zugrundeliegenden System (z.B. Hardware) wird in internen Servern gekapselt, die nur mit dem Mikrokernel kommunizieren. Externe Server kombinieren die atomaren Dienste des Mikrokernels zu komplexeren Funktionseinheiten. Adapter entkoppeln die Anwendungen von den externen Servern. So kann es z.B. unterschiedliche Adapter für Web- oder lokale Anwendungen geben. Die Anwendungen greifen nur auf die Adapter zu. [BMRSS96]

Obwohl er ursprünglich im Bereich der Betriebssysteme entstanden ist, kann dieser Ansatz auch für Anwendungsprogramme genutzt werden. Die Anwendung ist dann die Benutzeroberfläche, die Programmlogik steckt in den externen Servern und bedient sich elementarer Dienste aus dem Kernel. Der Kernel ist durch die internen Server vom Betriebssystem entkoppelt. Abbildung 11: Mikrokernel stellt eine solche Interpretation dieses Musters dar. Aufgrund dieser Anwendungsmöglichkeit ist Mikrokernel sehr interessant für eine Betrachtung bezüglich Evolvability.

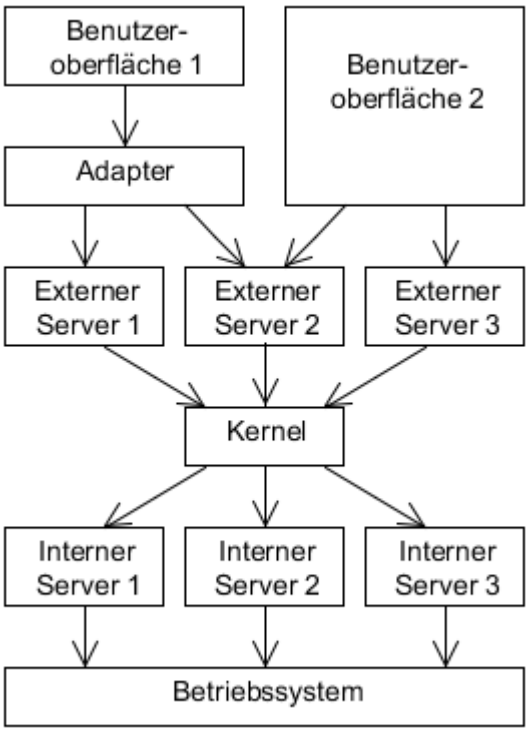


*Abbildung 11: Mikrokernel*

### Reflection

Die Implementierung der Software wird auf Basis von Metainformationen (Laufzeitinformationen) realisiert. Diese Metainformationen sind in einem eigenen Metalevel gespeichert, auf welches das Basislevel mit der eigentlichen Anwendung aufbaut. Ändern sich die Metainformationen, so erkennt dies die Software selbständig und reagiert entsprechend darauf. Die Metainformationen sind z.B. Informationen über die Speicherstruktur eines Objektes. So kann eine Persistenzkomponente jedes beliebige Objekt auf Basis dieser Metainformationen speichern. Änderungen an der Speicherstruktur der Objekte ziehen nun keine Änderungen an der Persistenzkomponente mehr nach sich. Gleiches ist auch für andere Aspekte, wie z.B. den Mechanismus für Funktionsaufrufe realisierbar. [BMRSS96]

### Plug-In

Ein Plug-In (engl. für hineinstecken, anschließen) ist eine Erweiterungsbibliothek für eine Software, welche zur Laufzeit eingebunden wird. Sie ist speziell für diese Software programmiert. [MVN06]

# 3 Vorgehen bei der Bewertung

Bei der Entwicklung einer Software führt die Auswahl geeigneter Muster nicht zwangsläufig zum Ziel, entscheidend ist auch deren Kombination zu einem mit den Projektzielen im Einklang stehenden Softwareentwurf und dessen Umsetzung. Daher wird bei der Bewertung grundsätzlich von einem sachgemäßen und vorteilhaften Einsatz der Muster ausgegangen. Schafft beispielsweise ein bestimmtes Muster gute Voraussetzungen für die Umsetzung eines bestimmten Prinzips, so wird das zu einer guten Bewertung bezüglich des Prinzips führen.

## *3.1 Zerlegung von Evolvability*

Die im Abschnitt Recherche genannten Muster sollen nach ihren Auswirkungen auf die Evolvability des daraus resultierenden Programms untersucht werden. Da Evolvability ein sehr umfassender Begriff ist, muss dieser zuerst in leichter fassbare Subcharakteristiken zerlegt werden, mittels dieser die Muster wesentlich intuitiver bewertet werden können. Dazu wurde [BBR09] als Grundlage verwendet. Im Anschluss werden die benutzten Subcharakteristiken für diese Arbeit definiert.

### 3.1.1 Definitionen

#### *Prinzipien und Attribute*

Der **Change Impact** drückt aus, wie groß der Aufwand für Änderungen an der Software ist.

Die **Kapselung** einer Komponente trennt das Interface der Komponente von seiner Implementierung. [BME07] Das heißt, dass eine Änderung in der Implementierung der Komponente unter Beibehaltung des Interfaces keine Auswirkung auf die Benutzer des Interfaces hat. Die Benutzer dürfen dazu keine (deshalb normalerweise unbekannten) Implementierungsdetails der Komponente ausnutzen, wenn sie ihr Interface benutzen. Mechanismen, die eine Umgehung der Schnittstellen erschweren, tragen zu einer guten Kapselung bei.

Die **Komplexität** einer Software steigt mit der Anzahl ihrer Komponenten und der Anzahl der Abhängigkeiten zwischen ihnen.

Hat ein Entwurf eine gute **Konsistenz**, so sind gleiche Probleme auf gleiche Art und Weise gelöst. Dazu gehört auch die Anwendung gleicher Muster auf gleiche Probleme.

**Kopplung** ist die Art und das Ausmaß der Abhängigkeit zwischen Softwarekomponenten. [IEEE 610.12]

**Korrektheit** bedeutet, dass die Komponente immer genau das zur Anfrage passende Ergebnis liefert. [SB03]

**Modularität** drückt aus, wie gut sich die Software in einzelne, klar abgegrenzte und leicht austauschbare Module unterteilen lässt. [GP70]

**Patency** drückt aus, inwiefern sich Artefakte aus den frühen Phasen des Entwurfs auf Artefakte der späteren Phasen des Entwurfs abbilden lassen. [BBR09]

**Separation of Concerns** ist die Aufteilung unterschiedlicher Zuständigkeiten auf unterschiedliche Komponenten.

**Typification** ist die Fähigkeit einer Komponente, in anderen Komponenten wiederverwendet zu werden oder als Teil einer Komponentenbibliothek Verwendung zu finden. [SB03] Das beinhaltet vor allem eine klare Definition und Dokumentation der Schnittstellen und angebotenen Services.

**Vollständigkeit** einer Komponente bedeutet, dass alle angebotenen Services auch nutzbar sind, incl. Spezifikation, Entwurf und Implementierung. [SB03]

#### *Qualitäten*

Die **Analysierbarkeit** spiegelt die Durchführbarkeit einer Analyse der Funktionalität einer Komponente wieder. Die Verständlichkeit der Analyse wird dadurch nicht beschrieben. Die Analysierbarkeit wird z.B. durch einen gut dokumentierten Entwurf unterstützt.

Hat eine Software eine gute **Änderbarkeit**, so sind gewünschte Modifikationen an dieser Software leicht zu implementieren. [ISO9126]

Die **Einhaltung von Standards** drückt aus, ob ggf. vorhandene, anwendbare Standards eingehalten werden.

Eine gute **Erweiterbarkeit** ist vorhanden, wenn sich die Komponente mit einem geringen Arbeitsaufwand an neue (speziell: erweiterte) Anforderungen anpassen lässt. [SB03]

Hat ein Entwurf eine gute **konzeptionelle Integrität**, so wurde auf mehrfach vorkommende, gleichartige Probleme mit den gleichen Mitteln und Lösungen eingegangen.

Die **Portability** einer Software gibt an, wie aufwendig es ist, sie in eine neue Umgebung einzubetten. [ISO9126]

Hat eine Software eine gute **Testbarkeit**, so sind Modifkationen an ihr leicht validierbar. [ISO9126]

**Traceability**, Rückverfolgbarkeit, stellt die Fähigkeit dar, die Entwicklungsschritte eines Systems durch die Verbindung der Requirements mit den Ergebnissen eines jeden Entwicklungsschrittes nachverfolgen und wiederherstellen zu können. [GF94]

**Variabilität** drückt aus, wie leicht ein Softwareentwickler, der eine Komponente benutzen möchte, erkennen kann, an welchen Stellen sie verändert werden kann, und wie leicht man den dazu notwendigen Mechanismus erkennen kann. [SB03] Dieses Attribut ist besonders im Zusammenhang mit Produktserien relevant.

Hat eine Software eine gute **Verständlichkeit**, so kann der Softwareentwickler schnell die Dokumentation, innere Struktur, Funktionsweise und Datenstrukturen der Software oder Komponente verstehen, so dass er diese anschließend ändern kann.

**Wiederverwendbarkeit** ist das Ausmaß, zu dem sich eine Softwarekomponente in mehr als einem Programm wiederverwenden lässt. [IEEE 610.12]

### 3.1.2 Hierarchie der Bewertungskriterien

Im Folgenden wird die in Abbildung 12: Hierarchie der Bewertungskriterien dargestellte Hierarchie vorgestellt und beschrieben.

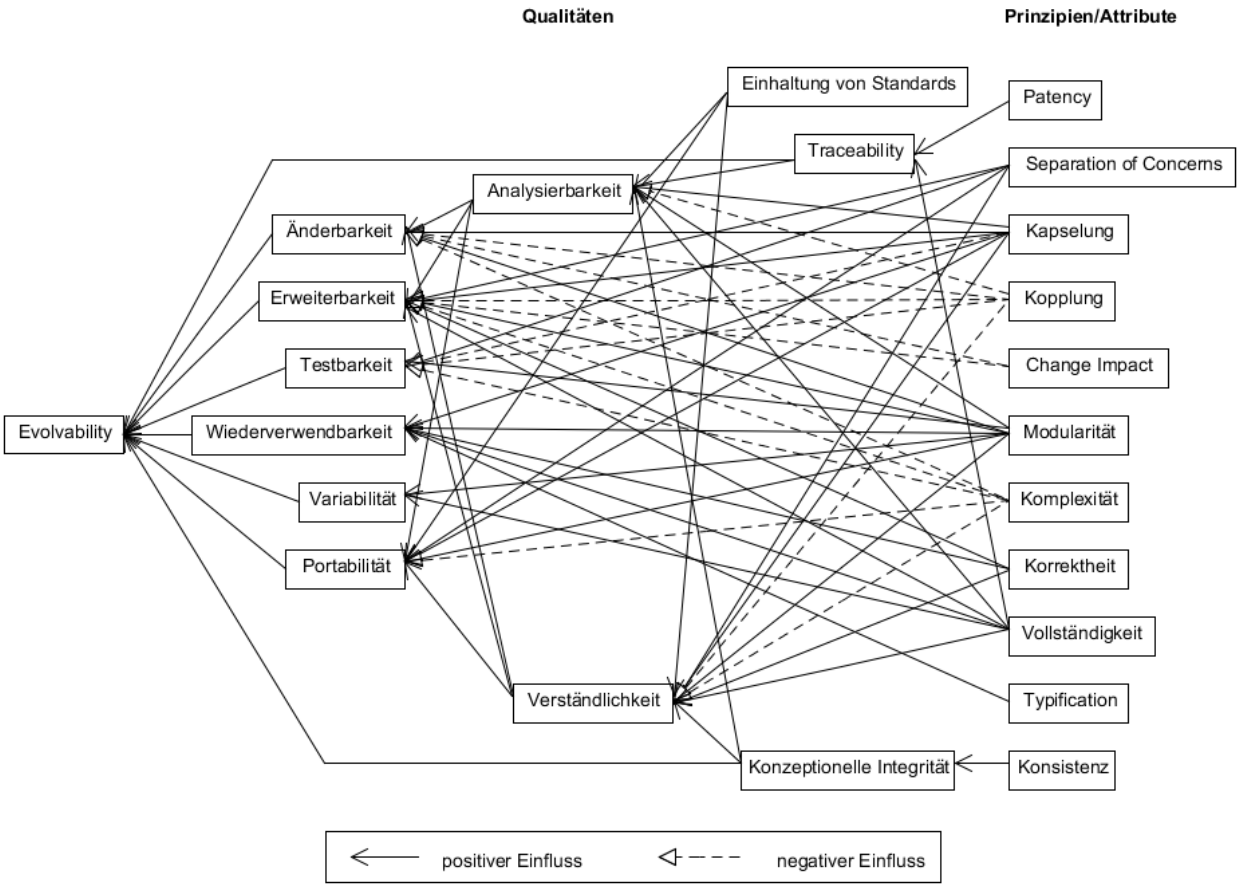


*Abbildung 12: Hierarchie der Bewertungskriterien*

Die Prinzipien und Attribute bilden die grundlegende Ebene der Hierarchie, da diese am leichtesten direkt zu bewerten sind. Die Qualitäten sind entsprechend ihrer Abhängigkeiten in den höheren Ebenen angeordnet, das Topziel ist die Evolvability. Es werden auch Kriterien berücksichtigt, die den Entwurf, dessen Umsetzung und die Dokumentation des Entwurfsprozesses betreffen. Die Hierarchie ist somit nicht auf die Bewertung von Mustern beschränkt, es können jegliche die Evolvability beeinflussenden Faktoren während der Entstehung einer Software damit bewertet werden. Um deren Einflüsse entlang der Abhängigkeiten zwischen den Elementen berechnen zu können, werden diese gewichtet. Im Folgenden werden die Einflüsse und die Verteilung der Gewichte beschrieben.

### *Erläuterung der Einflüsse*

#### Traceability

Vollständigkeit umfasst das Vorhandensein einer Dokumentation des Entwurfes und bei guter Patency ist diese so aufgebaut, dass man die Gründe der jeweiligen Entwurfsentscheidungen erkennen kann. Insgesamt ergibt sich daraus eine gute Traceability.

Die beiden Einflüsse sind nach Meinung des Autors gleich gewichtet.

#### Konzeptionelle Integrität

Konsistenz ist in hohem Maße für eine gute konzeptionelle Integrität verantwortlich.

#### Analysierbarkeit

Die Einhaltung von Standards bringt bekannte Strukturen in einen Entwurf, wodurch die Analyse vereinfacht wird. Gleiches trifft auf einen durch gute Traceability nachvollziehbaren Entwurf zu. Kapselung und Modularität bewirken eine Zerteilung des Projektes in mehrere wenig voneinander abhängige Teile, welche einzeln leichter analysierbar sind. Kopplung wirkt sich geradezu gegensätzlich aus. Die Vollständigkeit umfasst die Spezifikation der Schnittstellen und bildet somit eine verlässliche Arbeitsgrundlage. Konzeptionelle Integrität hat Vorteile, weil dadurch gleiche, bereits analysierte Lösungsansätze mehrfach vorkommen.

Vollständigkeit, Traceability und konzeptionelle Integrität werden gegenüber den anderen Einflüssen in doppeltem Maße gewichtet.

#### Verständlichkeit

Aus den gleichen Gründen wie bei der Analysierbarkeit wirken sich die Einhaltung von Standards, konzeptionelle Integrität, Kapselung und Modularität sowie die Vollständigkeit auch auf die Verständlichkeit positiv aus. Durch Separation of Concerns sind verschiedene Zuständigkeiten in verschiedenen Komponenten realisiert, was der Übersicht dienlich ist und so der Verständlichkeit zuträgt. Eine hohe Kopplung oder Komplexität bewirkt genau das Gegenteil. Eine Grundvoraussetzung für die Verständlichkeit ist die Korrektheit des betrachteten Teiles der Software.

Komplexität, Korrektheit und Vollständigkeit wirken sich besonders auf die Verständlichkeit aus und gehen in dieser Arbeit deshalb im doppelten Maße in die Bewertung ein.

#### Änderbarkeit

Um einen Teil einer Software ändern zu können, muss sie zuerst analysiert und verstanden werden. Für die Umsetzung der Änderungen sind eine gute Kapselung und Modularität dienlich, da dadurch geringere Auswirkungen auf benachbarte Komponenten auftreten. Hohe Kopplung oder ein hoher Change Impact bewirken genau das Gegenteil. Eine hohe Komplexität bedeutet eine große Anzahl an abhängigen Komponenten.

Da die Einflüsse von Kapselung, Komplexität, Kopplung und Modularität insgesamt etwa gleich wichtig erscheinen wie die einzelnen Einflüsse von Change Impact, Analysierbarkeit und Verständlichkeit, werden die letztgenannten mit vierfachem Gewicht in die Bewertung der Änderbarkeit eingehen.

#### Erweiterbarkeit

Ebenso wie bei Änderungen, müssen bei Erweiterungen die bereits bestehenden Teile der Software beachtet werden. Daher wirken sich Analysierbarkeit, Verständlichkeit, Kapselung, Kopplung, Change Impact, Modularität und Komplexität aus den gleichen Gründen wie bei der Änderbarkeit auch auf die Erweiterbarkeit aus. Separation of Concerns hilft, den geeigneten Ansatzpunkt für die Erweiterung zu finden. Korrektheit und Vollständigkeit erleichtern die Integration.

Analysierbarkeit und Verständlichkeit werden sich in dieser Arbeit mit doppeltem, Modularität und Vollständigkeit mit vierfachem Gewicht auf die Bewertung der Erweiterbarkeit auswirken.

#### Testbarkeit

Soll ein Sachverhalt getestet werden, so ist es wenig dienlich, wenn sich die entsprechende Funktionalität über die gesamte Software verteilt, weshalb sich die Separation of Concerns positiv auswirkt. Eine Komponente lässt sich am einfachsten testen, wenn sie vollkommen selbständig nutzbar ist. Deshalb wirken sich die Modularität positiv, sowie Kopplung und Komplexität negativ aus. Ist eine Komponente stark gekapselt, so ist sie nur schwer beobachtbar, was eine Voraussetzung für das Testen ist.

Alle Abhängigkeiten werden gleich gewichtet.

#### Wiederverwendbarkeit

Gute Typification und Vollständigkeit bedeuten unter anderem das Vorhandensein einer klaren Schnittstellenspezifikation, welche sich sehr positiv auf die Wiederverwendbarkeit auswirkt. Eine gute Kapselung trennt die Schnittstelle von ihrer Implementierung, wodurch bei einer Wiederverwendung der Komponente lediglich die Schnittstelle von Interesse ist. Je besser die Modularität ist, desto geringer fällt der Aufwand bei einer Wiederverwendung aus. Arbeitet eine Komponente nicht entsprechend ihrer Schnittstellenspezifikation (Korrektheit), so macht es wenig Sinn, sie in weiteren Projekten wiederzuverwenden.

Typification und Vollständigkeit werden jeweils mit doppeltem Gewicht in die Bewertung der Wiederverwendbarkeit eingehen.

#### Variabilität

Entscheidend für die Variabilität sind vor allem die Einstellmöglichkeiten der Komponente. Auch eine gute Modularität und Vollständigkeit wirken sich positiv aus, da sich dadurch leichter neue Kombinationen von Komponenten bilden lassen.

Die Abhängigkeiten werden alle gleich gewichtet.

#### Portabilität

Soll eine Komponente in eine neue Umgebung eingebettet werden, so müssen gegebenenfalls Änderungen vorgenommen werden, die eine gute Analysierbarkeit sowie Verständlichkeit voraussetzen. Durch die Einhaltung von Standards erhöht sich die Wahrscheinlichkeit, dass vorhandene Schnittstellen bei einer Portierung keiner Anpassung bedürfen oder entsprechende Adapter bereits existieren. Separation of Concerns, Kapselung und Modularität sind von Vorteil, um die anzupassenden Elemente finden und ändern zu können. Eine hohe Komplexität wirkt sich genau darauf hinderlich aus.

Separation of Concerns wird doppeltes, die Modularität dreifaches Gewicht haben.

#### Evolvability

Entwickelt sich eine Software weiter, muss die konzeptionelle Integrität gewahrt bleiben, die Traceability muss weiter gepflegt werden, es müssen leicht Änderungen und Erweiterungen vorgenommen werden können und sie muss testbar bleiben. Desweiteren sind die Variabilität, Portabilität und Wiederverwendbarkeit wichtige Teile der Evolvability [BBR09].

Änderbarkeit, Erweiterbarkeit und Portability werden mit doppeltem Gewicht in die Bewertung der Evolvability eingehen, da diese nach Meinung des Autors die Kernqualitäten darstellen.

## *3.2 Bewertungsskala*

Jedes Muster wird nach seiner Auswirkung auf jedes Prinzip, jedes Attribut und jede Qualität auf einer Skala von -3 bis +3 bewertet. Wirkt sich das Muster hemmend auf das Kriterium aus, wird negativ bewertet. Wirkt es sich unterstützend aus, gibt es eine positive Bewertung. Beeinflusst das betrachtete Muster das Kriterium nicht, so wird indifferent mit 0 bewertet. Ein Betrag von 1 bedeutet nur leichten Einfluss, 3 einen besonders starken Einfluss.

Bewirkt der Einsatz eines Musters beispielsweise eine erhebliche Erhöhung der Komplexität, wird trotz der negativen Auswirkungen hoher Komplexität der Einfluss auf die Komplexität selbst positiv mit einer +3 bewertet. Da sich hohe Komplexität z.B. negativ auf die Verständlichkeit auswirkt, wird die Bewertung der Komplexität entsprechend mit einem negativen Gewicht in die Bewertung der Verständlichkeit eingehen.

## *3.3 Bewertbarkeit der Kriterien*

Die in Kapitel 3.1 - Zerlegung von Evolvability vorgestellte Hierarchie ist für die Bewertung diverser Architekturlösungsansätze geeignet und enthält dementsprechend eine umfassende Vielfalt an Attributen, Prinzipien und Qualitäten, die sich auf die Evolvability auswirken. Nicht jedes dieser Kriterien kann aber von Mustern beeinflusst werden. In diesem Kapitel wird diskutiert, inwiefern ein Einfluss möglich ist und wie dieser bewertet werden kann. Ist diese Möglichkeit nicht gegeben, wird das entsprechende Kriterium indifferent mit Null bewertet.

### *Prinzipien und Attribute*

#### Change Impact

Wie weit die Auswirkungen einer Änderung an einer Komponente der betrachteten Software reichen, liegt an der Kapselung dieser Komponente, falls sich die Änderung nicht auf die Schnittstelle der Komponente auswirkt. Ist die Kapselung gut, gibt es keine

Auswirkungen auf andere Komponenten. Wirkt sich die Änderung auf die Schnittstelle aus, so hängt es von der Komplexität der Konstruktion der Software ab, wie viele andere Komponenten davon betroffen sind. Bei der Bewertung des Change Impacts werden Einflüsse von Mustern berücksichtigt, die den Umfang der notwendigen Änderungen am Quelltext bei der Umsetzung von Änderungen verringern.

#### Kapselung

Eine optimale Trennung von Schnittstelle und Implementierung ist durch ein Verstecken der nicht schnittstellenrelevanten Funktionalität möglich. Durch deren Definition in privaten Eigenschaften, Methoden und Ereignissen, wird der Zugriff von außerhalb durch die Programmiersprache verhindert. Somit ist das Verstecken ein Implementierungsdetail. Bietet die genutzte Programmiersprache keine Möglichkeit Funktionalität zu verstecken, so liegt es am Programmierer, die interne Funktionalität von außerhalb der Komponente nicht direkt zu nutzen und keine Rückschlüsse darauf zu ziehen. Auch in diesem Fall ist Kapselung also eine Frage der Implementierung. Bei der Bewertung wird grundsätzlich von einer guten Kapselung bei der Implementierung ausgegangen, um in die Bewertung keine Folgen schlechter Programmierung eingehen zu lassen. Hier werden demnach vor allem Einflüsse bewertet, die eine Umgehung der Schnittstelle erschweren.

#### Komplexität

Die Bestimmung der Anzahl der Komponenten eines Softwaresystems sollte zuerst über die Anzahl der klar trennbaren Zuständigkeiten erfolgen. Diese Anzahl wird bei der Bewertung der Komplexität als gegeben angesehen. Eine minimale Komplexität (Bewertung -3) entsteht, wenn es pro Komponente nur eine Abhängigkeit gibt (z.B. Sterntopologie). Das Maximum (Bewertung +3) entsteht, wenn jede Komponente von jeder anderen abhängig ist. Zusätzliche Komponenten, die durch die Nutzung eines Musters notwendig werden, werden ebenso als Komplexitätszunahme gewertet.

#### Konsistenz

Die Konsistenz bewertet das Zusammenspiel mehrerer Muster in einem Entwurf. Da hier nur einzelne Muster bewertet werden, kann folglich zur Konsistenz des Entwurfs keine Aussage gemacht werden.

#### Kopplung

Wenn Muster eine Abhängigkeit zwischen Komponenten definieren, so ist es meist dem Softwarearchitekten oder gar dem Programmierer überlassen, welche Art von Kopplung letztlich implementiert wird. In diesem Fall wird indifferent mit Null bewertet. Schreibt ein Muster jedoch eine entkoppelnde Vorgehensweise vor, so wird das entsprechend mit negativem Betrag bewertet.

#### Korrektheit

Arbeitet eine Software nicht korrekt, so ist das auf einen inhaltlichen Fehler in der Funktionalität oder auf einen Implementierungsfehler zurückzuführen. Für beides sind Muster nicht zuständig.

#### Modularität

Modularität umfasst klar definierte Schnittstellen, die nicht durch die Implementierung umgangen werden. Dies sicherzustellen ist eine Frage der Umsetzung, nicht der Muster. Bewertet wird hier, inwieweit Muster eine strukturelle Separierung in Module erleichtern oder vorschreiben.

#### Patency

Um eine Abbildung von Artefakten verschiedener Entwurfsphasen aufeinander zu ermöglichen, ist eine entsprechende Dokumentation des Vorgehens beim Entwurf nötig. Muster haben keinen Einfluss darauf, ob eine solche Dokumentation während des Entwurfes erstellt wird.

#### Separation of Concerns

Sofern ein Muster eine inhaltliche Separierung erleichtert oder vorschreibt, wird dies zu einer positiven Bewertung führen. Die Bewertung ist allerdings kein Garant für eine tatsächliche Separierung in der Umsetzung der Muster, sie spiegelt lediglich den notwendigen Aufwand dafür wieder.

#### Typification

Ob eine Komponente in einer Bibliothek Verwendung finden kann, kommt auf die Anforderungen der Bibliothek an, welche bei einer Beurteilung dieser Möglichkeit bekannt

sein müssten. Die Wiederverwendung in anderen Komponenten hängt entsprechend davon ab, ob passende (standardisierte) Schnittstellen existieren. Die Verwendung solcher Schnittstellen wird in keinem der in dieser Arbeit untersuchten Muster vorgeschrieben. Aus diesen Gründen kann keine Bewertung für die Typification vorgenommen werden.

#### Vollständigkeit

Die angebotenen Services werden durch die Schnittstellenspezifikation definiert. Sind sie nicht nutzbar, so liegt das an einer nicht ausreichenden Dokumentation oder unzureichenden Implementierung. Für beides sind Muster nicht zuständig.

### *Qualitäten*

Die Bewertung der Qualitäten wird sich zum größten Teil aus den Abhängigkeiten ergeben, wie sie im Kapitel 3.1 - Zerlegung von Evolvability vorgestellt wurden. Ergibt sich aus dem Kontext eines Musters ein Einfluss, der durch diese Abhängigkeiten nicht oder nicht in entsprechendem Ausmaß berücksichtigt wird, so wird diese Qualität entsprechend direkt bewertet. In der Regel wird solch eine direkte Bewertung allerdings nicht notwendig sein. Da einige der Prinzipien und Attribute nicht bewertet werden können, wird der Betrag der Bewertung der Qualitäten in einigen Fällen gering ausfallen. Das liegt daran, dass für diese Qualitäten auch die Umsetzung der Muster im Entwurf sowie in der Implementation der Software von hoher Bedeutung sind.

#### Portability

Die Portability ist sehr abhängig von der Verfügbarkeit eines Compilers der richtigen Programmiersprache für das neue Betriebssystem, sowie von der Kompatibilität der verwendeten Bibliotheken. All das sind aber Faktoren, die vielmehr die Umsetzung betreffen, als die Muster, die in einem Projekt verwendet werden. Werden allerdings Teile der Software, die bei einer Portierung besonders oft von Änderungen betroffen sind, in eigenen Komponenten gekapselt, so kann das als Vorteil bewertet werden.

#### Variabilität

Durch ihre starke Abhängigkeit von Faktoren, die durch die gewählten Prinzipien und Attribute nicht erfasst werden können, wie zum Beispiel die Verwendung von zentralen Konstanten statt festen Werten im Quelltext, kann die Variabilität nur in geringem Maße durch Abhängigkeiten bewertet werden. Da die meisten dieser Faktoren nicht von Mustern beeinflusst werden, wird der Betrag der Bewertung der Variabilität meist gering ausfallen.

# 4 Bewertung der Muster

In diesem Kapitel werden die Einflüsse der Muster auf die relevanten Kriterien der in Kapitel 3.1.2 - Hierarchie der Bewertungskriterien vorgestellten Hierarchie ermittelt. Dazu wird jedes Muster exemplarisch auf eines der beiden nachfolgend vorgestellten Musterbeispiele angewendet und bewertet.

Die Bewertungen spiegeln lediglich die Auswirkungen auf den Einflussbereich des jeweiligen Musters wieder.

## *4.1 Die Musterbeispiele*

Um die Bewertung der Muster zu erläutern, werden sie jeweils auf einen passenden Ausschnitt eines der im Folgenden angeführten Beispiele angewandt.

### 4.1.1 Beispiel 1: Verwaltungsprogramm für Sammelbesteller

Versandhäuser arbeiten gerne mit Sammelbestellern zusammen, die von mehreren Kunden Bestellungen annehmen und diese als Sammelbestellung beim Versandhaus einreichen. Das Versandhaus liefert die Waren in einer Sendung an den Sammelbesteller aus, der diese wiederum an seine Kunden verteilt. Das hat folgende Vorteile: Der Sammelbesteller kennt Formalitäten und Abläufe bei selten auftretenden Vorgängen wie Rücksendungen, Reklamationen, Ratenzahlungen usw. besser als der Endkunde. So hat der Mitbesteller einen persönlichen, vertrauten Ansprechpartner und das Versandhaus delegiert die immer gleiche Aufklärungsarbeit an den Sammelbesteller.

Als Beispiel soll eine Software für Sammelbesteller dienen. Sie sollen damit Bestellungen aufgeben, ihre Mitbesteller verwalten und Reklamationen tätigen können. Bei der Betrachtung der einzelnen Muster soll jeweils ein passender Teil dieser Software entworfen werden, um das Muster und dessen Bewertung besser erläutern zu können.

### 4.1.2 Beispiel 2: Aufnahme eines Videos

Ein Video soll von einer analogen Videoquelle aufgenommen, komprimiert und digital gespeichert werden. Die Daten der analogen Videoquelle werden vom Betriebssystem in digitaler Form uncodiert zur Verfügung gestellt. Bei der Verarbeitung der Videodaten kann sich wahlweise eines Multimediaframeworks bedient werden.

## *4.2 Architekturstile*

#### *Client-Server*

Die Bestellungen, die der Sammelbesteller in seinem Programm eingibt, müssen in regelmäßigen Abständen an das Versandhaus übermittelt werden. Das geschieht bisher per Telefon. Im Callcenter werden die Bestellungen in ein Programm des Versandhauses eingegeben. Viel effizienter ist natürlich eine direkte Übertragung der Bestelldaten vom Programm des Sammelbestellers zum Programm des Versandhauses über das Internet. Für derartige Aufgaben hat sich seit Jahrzehnten Client-Server etabliert.

Das Versandhaus installiert zu diesem Zweck einen Server, der aus dem Internet erreichbar ist. Dieser stellt eine Schnittstelle für die Übertragung der Bestellungen zur Verfügung. Das Programm der Sammelbesteller kann nun als Client Kontakt aufnehmen und die Bestellungen übertragen. Der Aufbau des Servers ist genau wie der des Clients nicht weiter spezifiziert, hier kommen weitere Muster zur Anwendung. Vorgeschrieben ist lediglich, dass nur der Client eine Verbindung aufbauen kann, der Server muss sich passiv verhalten. Er darf lediglich eine entsprechende Antwort auf die Anfrage vom Client an diesen zurückschicken. Abbildung 13: Beispiel für Client-Server verdeutlicht die Anordnung.

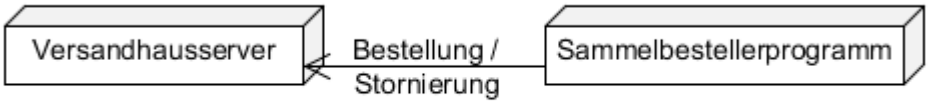


*Abbildung 13: Beispiel für Client-Server*

Client-Server enthält keinerlei Vorschriften, die den **Change Impact** beeinflussen: **0**

Es mag naheliegen, Client-Server eine positive Bewertung bezüglich der Kapselung zu geben, weil schließlich eine Verbindung über das Internet aufgebaut wird und durch die dabei notwendige Datencodierung schon eine Trennung von Implementierung und Schnittstelle entsteht. Doch diese Art der Verwendung ist weder zwingend vorgeschrieben, noch ist sie die einzig mögliche. Beispielsweise entsprechen auch die COM-Dienste unter Windows dem Client-Server Stil und stellen damit ein Beispiel einer lokalen Implementierung dar. Da also Client-Server keinerlei direkte oder indirekte Vorschriften zur Trennung von Interface und Implementierung macht, kann die Bewertung der **Kapselung** auch nur indifferent **0** betragen.

Der Versandhausserver ist nicht nur für einen Sammelbesteller ausgelegt, sondern für tausende. Jeder Client hat nur eine Abhängigkeit: Den Versandhausserver. Das ist die minimal mögliche **Komplexität**: **-3**

Typischerweise wird der Server über eine URL kontaktiert (z.B. bestellungen.versandhaus.de), die vorher über einen DNS-Server in eine IP-Adresse aufgelöst wird und somit vom Server entkoppelt. Da aber Client-Server weder diesen noch einen anderen Mechanismus zur Entkopplung vorschreibt, kann die Bewertung der Kopplung nicht negativ ausfallen. Da auch sonst keine die **Kopplung** erhöhenden Vorschriften gemacht werden, kann die Bewertung nur indifferent **0** lauten.

Da zwischen dem Server und den Clients eine klar definierte Schnittstelle besteht, kann es problemlos verschiedene Versionen der Clients geben, solange sich alle an die gleiche Schnittstellendefinition halten. Daher bekommt Client-Server für die **Modularität** eine **+3**.

Der Versandhausserver nimmt die Bestellungen entgegen und stellt sie weiteren verarbeitenden Systemen zur Verfügung. Der Client beim Sammelbesteller ist dagegen für die Verwaltung von Bestellungen und deren gebündelte Übertragung an das Versandhaus sowie für die Datenhaltung der Kundendaten zuständig. Da der Server keine Verbindungen initiieren kann (also niemals als Client agieren kann), ist eine gewisse **Separation of Concerns** durch den Stil vorgegeben: **+1**

Bei Client-Server Systemen ist eine Erhöhung der Anzahl der Clients im Normalfall problemlos möglich, soweit dies nicht durch andere Einschränkungen verhindert wird. Die einzige Grenze ist hierbei die Rechen- und Kommunikationskapazität des Servers. Dieser Stil hat somit eine positive Auswirkung auf die quantitative **Erweiterbarkeit**, welche folglich mit einer **+2** bewertet wird.

Tabelle 1: Bewertung von Client-Server zeigt eine übersichtliche Zusammenfassung.

| **Prinzipien und Attribute** | **Bewertung** | **Qualitäten** | **Bewertung** |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | 0 | Änderbarkeit | 0 |
| Komplexität | -3 | Erweiterbarkeit | +2 |
| Kopplung | 0 | Portability | 0 |
| Modularität | +3 | Testbarkeit | 0 |
| Separation of Concerns | +1 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 1: Bewertung von Client-Server*

### *Layers, Tiers*

Layers eignet sich z.B. zur Strukturierung des Clients. Dabei implementiert das oberste Layer die Benutzerinteraktion des Programms mit dem Sammelbesteller, also die graphische Benutzeroberfläche. Diese stützt sich auf das darunterliegende Layer, welches die gesamte Programmlogik kapselt. Dazu gehören z.B. die Berechnung von Ratenzahlplänen, eine Suche nach bestellten, aber noch nicht eingetroffenen Artikeln, um diese bei Lieferung auch im Programm als geliefert markieren zu können oder ein Erinnerungsmechanismus, um die Rückgabefrist nicht zu überschreiten. Die unterste Schicht stellt grundlegende Funktionen wie z.B. die konsistente Speicherung von Daten in einer Datenbank oder in Dateien und die Kommunikation mit dem Versandhausserver bereit. Die Kommunikation zwischen den einzelnen Schichten gleicht der von Client-Server, wobei immer die untere Schicht die Serverschnittstelle und die darüberliegende Schicht die Clientschnittstelle implementiert. Ausnahmen, wie sie z.B. für den Erinnerungsmechanismus notwendig sind (hier muss die mittlere Schicht eine Verbindung mit der oberen Schicht initiieren, wenn die Zeit abgelaufen ist), sind zwar möglich, entsprechen aber nicht der ursprünglichen Schichtenarchitektur. Abbildung 14: Beispiel für Layers stellt die Anordnung dar.

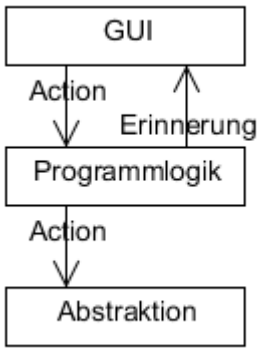


*Abbildung 14: Beispiel für Layers*

Dieser Stil enthält keinerlei Vorschriften, die den **Change Impact** oder die **Kapselung** beeinflussen: **0**

Wie im Strukturdiagramm in Abbildung 14: Beispiel für Layers zu sehen ist, hat jede Komponente (ausgenommen der untersten) nur jeweils eine Verbindung zu der darunterliegenden Schicht. Das entspricht der geringsten möglichen **Komplexität**: **-3** Lediglich die zusätzliche nicht ganz vorschriftsmäßige Verbindung von der Logik- in die GUI-Schicht für die Erinnerungsfunktion verursacht eine Erhöhung der Komplexität.

Da der Kommunikationsmechanismus dem von Client-Server gleicht, ist der Einfluss auf die **Kopplung** ebenfalls **0**.

Die Unterteilung in GUI-, Logik- und Basisschicht und die klar definierten Schnittstellen dazwischen, fordern eine klare strukturelle Unterteilung, was die **Modularität** sehr fördert: **+3**

Wie bei Client-Server kann jede Schicht nur Verbindungen in die nächsttiefere Schicht initiieren. Dadurch ergibt sich eine Hierarchie von Diensten, die sich auf die einzelnen Schichten verteilen. Die GUI nutzt die Dienste der Logik, welche wiederum auf die Basisschicht zurückgreift, um grundlegende Dienste, wie die konsistente Speicherung von Daten, in Anspruch zu nehmen. So ergibt sich automatisch eine **Separation of Concerns**, was mit einer **+2** bewertet wird.

Durch die Existenz eigener Komponenten für GUI und Abstraktion von der Umgebung wird die **Portability** unterstützt: **+2**

Tabelle 2: Bewertung von Layers, Tiers zeigt eine übersichtliche Zusammenfassung.

| Prinzipien und Attribute | Bewertung | Qualitäten | Bewertung |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | 0 | Änderbarkeit | 0 |
| Komplexität | -3 | Erweiterbarkeit | 0 |
| Kopplung | 0 | Portability | +2 |
| Modularität | +3 | Testbarkeit | 0 |
| Separation of Concerns | +2 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 2: Bewertung von Layers, Tiers*

***Repository***

Der primäre Einsatzzweck der Sammelbestellersoftware ist die Organisation und Verwaltung der Adress- und Bestelldaten der Mitbesteller. Es liegt also nahe, als zentrales Element eine Datenbasis einzusetzen und den Rest der Software nach dem Stil Repository rundherum aufzubauen. Die Prozesse sind z.B. das Anlegen oder Ändern von Mitbestellern, das Erfassen von Bestellungen oder Aktualisieren der Lieferadresse. Der Prozess zum Anlegen eines Kunden erstellt einen neuen leeren Datensatz und ruft direkt den Prozess zum Ändern auf, damit der Benutzer die Adressdaten eingeben kann. Der gleiche Aufbau kann für die Verwaltung der Bestelldaten eingesetzt werden. In Abbildung 15: Beispiel für Repository ist der Zusammenhang der Komponenten dargestellt.

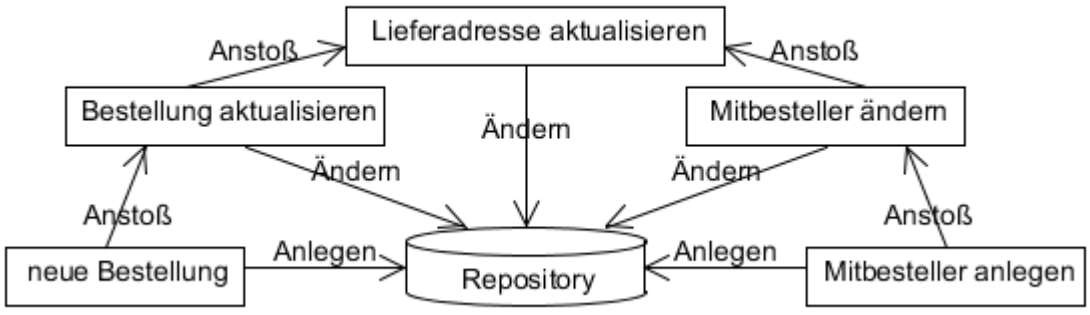


*Abbildung 15: Beispiel für Repository*

Dieser Stil enthält keinerlei Vorschriften, die den **Change Impact** oder die **Kapselung** beeinflussen: **0**

Die Komplexität, die durch die Verbindungen der um das Repository herum angeordneten Prozesse (z.B. Anlegen und Ändern der Adressdaten) untereinander entsteht, wird durch das Repository nicht beeinflusst. Allerdings stellt das Repository selbst eine weitere Komponente dar, zu der auch eigene Verbindungen bestehen. Dadurch erhöht sich die **Komplexität** des gesamten Systems etwas: **+1**

Über die **Kopplung** zwischen den Prozessen und dem Repository werden keine Vorschriften gemacht, daher wird indifferent mit **0** bewertet.

Per Definition gibt es eine eigene zentrale Komponente, die nur für die Datenspeicherung zuständig ist. Das ergibt für die **Separation of Concerns** eine **+2**. Die restlichen Zuständigkeiten können je nach Implementierung mehr oder weniger gut auf einzelne Komponenten aufgeteilt werden. Dadurch können relativ viele Abhängigkeiten zur zentralen Komponente entstehen, die bei einem Austausch der Datenhaltungskomponente alle aktualisiert werden müssen, so dass die **Modularität** nur leicht unterstützt wird: **+1**

Tabelle 3: Bewertung von Repository zeigt eine übersichtliche Zusammenfassung.

| **Prinzipien und Attribute** | **Bewertung** | **Qualitäten** | **Bewertung** |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | 0 | Änderbarkeit | 0 |
| Komplexität | +1 | Erweiterbarkeit | 0 |
| Kopplung | 0 | Portability | 0 |
| Modularität | +1 | Testbarkeit | 0 |
| Separation of Concerns | +2 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 3: Bewertung von Repository*

***Blackboard***

Zur Verbildlichung des Blackboard-Stils lässt sich das gleiche Szenario wie beim Repository anwenden. Der Unterschied ist lediglich, dass sich die Prozesse bei Programmstart selbständig bei der Datenhaltungskomponente (dem Blackboard) registrieren. Dies ist in Abbildung 16: Beispiel für Blackboard dargestellt.

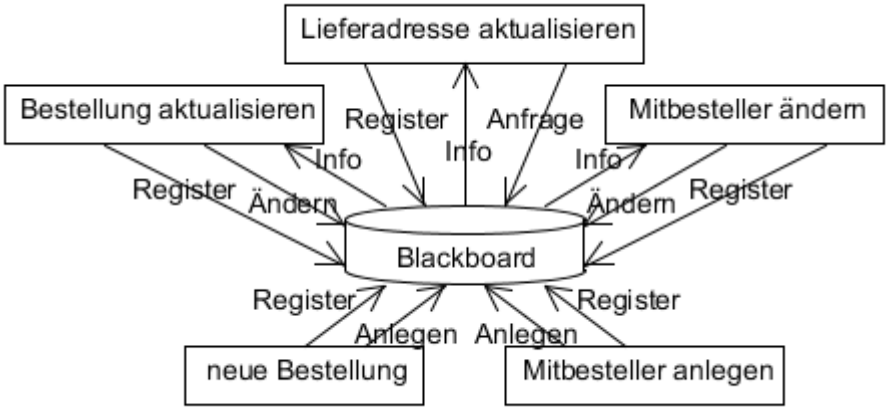


*Abbildung 16: Beispiel für Blackboard*

Die Bewertung von Blackboard gleicht der des Repositories bis auf nachfolgende Änderungen:

Der Prozess zum Ändern der Adressdaten wird bei der Registrierung angeben, dass er aufgerufen werden will, wenn ein neuer Mitbesteller angelegt wurde. Nun muss der Prozess zum Anlegen neuer Mitbesteller den Prozess zum Ändern der Daten nicht mehr explizit aufrufen. Hier entfällt eine Anhängigkeit. Auf äquivalente Weise verschwinden beim Blackboard im Vergleich zum Repository eine Großzahl der Abhängigkeiten zwischen den Prozessen, die mit der Datenbasis kommunizieren. Das wirkt sich hemmend auf die **Komplexität** aus: **-2**

Die eigenständige Registrierung der Prozesse bei der Datenbasis stellt ein Entkopplungsmechanismus dar. Die Datenbasis braucht nun bei einem Austausch von Prozessen nicht mehr geändert werden. Dieser einseitige Mechanismus bewirkt eine leicht hemmende Wirkung auf die **Kopplung**: **-1**

Tabelle 4: Bewertung von Blackboard zeigt eine übersichtliche Zusammenfassung.

| Prinzipien und Attribute | Bewertung | Qualitäten | Bewertung |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | 0 | Änderbarkeit | 0 |
| Komplexität | -2 | Erweiterbarkeit | 0 |
| Kopplung | -1 | Portability | 0 |
| Modularität | +1 | Testbarkeit | 0 |
| Separation of Concerns | +2 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 4: Bewertung von Blackboard*

### *Batchprogramme*

Es sind in einem Computersystem mehrere einzelne Programme verfügbar, die jeweils eine der geforderten Aufgaben bei der Aufnahme eines digitalen Videos bewerkstelligen können. Das erste Programm kommuniziert mit der Videoquelle (z.B. Grafikkarte mit Videoeingang oder TV-Karte) und speichert die Videodaten uncodiert in einer Datei. Ein zweites Programm schneidet schwarze Balken ab und führt eine Rauschunterdrückung durch. Ein drittes Programm codiert das Video auf eine akzeptable Größe. Ein Batchprogramm führt die 3 Programme automatisch nacheinander aus, wie in Abbildung 17: Beispiel für Batchprogramm dargestellt ist.

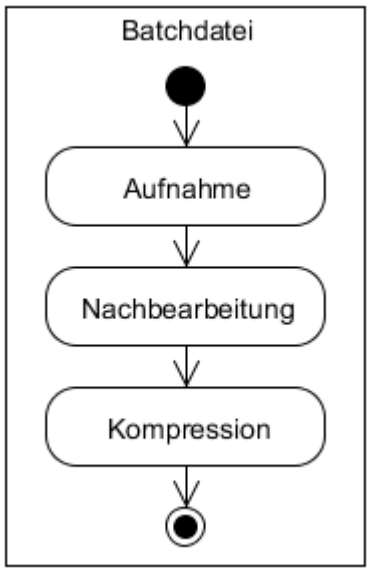


*Abbildung 17: Beispiel für Batchprogramm*

Dieser Stil beeinflusst den **Change Impact** nicht: **0**

Dadurch, dass die 3 Programme in eigenen Adressbereichen und außerdem nacheinander ausgeführt werden, ist es praktisch unmöglich, das Interface zwischen den Programmen (in diesem Fall die Daten in der Datei) zu umgehen. Somit ist die **Kapselung** extrem stark ausgeprägt: **+3**

Das Batchprogramm selbst ist nur eine Textdatei, die nacheinander die auszuführenden Programme auflistet. Die **Komplexität** ist daher minimal: **-3**

Die Programme sind komplett unabhängig voneinander, sie müssen lediglich das verwendete Dateiformat verstehen, eine weitere **Kopplung** besteht prinzipbedingt nicht: **-3**

Aus den gleichen Gründen ist die **Modularität** maximal ausgeprägt (Bewertung: **+3**), jedes Programm kann problemlos gegen ein gleichwertiges ausgetauscht werden.

Inwiefern einzelne Zuständigkeiten auf separate Programme aufgeteilt werden, wird durch den Stil in keiner Weise vorgeschrieben. **Separation of Concerns**: **0**

Aufgrund der Einfachheit im Aufbau eines Batchprogrammes, welches meist eine einfache Abfolge von Programmaufrufen in einer Textdatei ist, werden die **Analysierbarkeit**, **Erweiterbarkeit** und **Verständlichkeit** mit **+2** und die **Änderbarkeit** mit **+3** bewertet.

Tabelle 5: Bewertung von Batchprogramme zeigt eine übersichtliche Zusammenfassung.

| Prinzipien und Attribute | Bewertung | Qualitäten | Bewertung |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | +2 |
| Kapselung | +3 | Änderbarkeit | +3 |
| Komplexität | -3 | Erweiterbarkeit | +2 |
| Kopplung | -3 | Portability | 0 |
| Modularität | +3 | Testbarkeit | 0 |
| Separation of Concerns | 0 | Variability | 0 |
| | | Verständlichkeit | +2 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 5: Bewertung von Batchprogramme*

### *Pipes and Filters*

Das Multimediaframework, welches bei der Aufnahme des analogen Videos benutzt wird, setzt den Pipes and Filters Stil um. Die Filterkette sieht dann wie in Abbildung 18: Beispiel für Pipes and Filters aus.

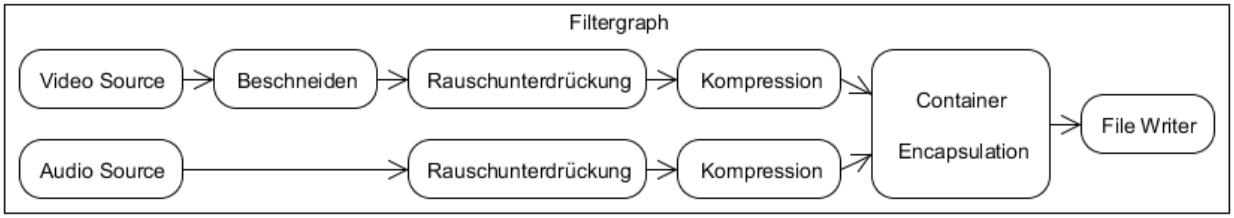


*Abbildung 18: Beispiel für Pipes and Filters*

Es existieren zwei Eingabefilter, einer für den Ton und einer für das Bild. Das Bild wird zuerst beschnitten, dann wird auf beide Streams eine Rauschunterdrückung angewendet, bevor sie codiert werden. Anschließend werden sie in einen Container gepackt, der z.B. für die Synchronität verantwortlich ist, welcher anschließend in eine Datei geschrieben wird.

Pipes and Filters beeinflusst den **Change Impact** nicht: **0**

Jeder Filter stellt eine in sich abgeschlossene Funktionalität bereit. Dadurch können aus mehreren Filtern viele verschiedene Filterketten zusammengesetzt werden. Beispielsweise könnte man in der Filterkette im Beispiel problemlos zuerst die Rauschunterdrückung anwenden und danach erst zurechtschneiden. Da die Filter also weder ihren Vorgänger, noch ihren Nachfolger kennen, müssen sie sich vollständig auf ihr Interface verlassen, was sich positiv auf die **Kapselung** auswirkt: **+2**

Filterketten sind, wie der Name schon sagt, in der Regel eine schleifenfreie Aneinanderreihung von Filtern. Jeder Filter hat dann im Normalfall genau eine Verbindung zum Vorgänger, was dem Mindestmaß an Komplexität entspricht. Da der Stil selbst Schleifen nicht ausschließt und diese zu einer höheren **Komplexität** führen, wird nicht auf minimale Komplexität, sondern mit **-2** bewertet.

Pipes and Filters beinhaltet keine direkten Vorschriften zur (Ent-)**Kopplung** der Filter: **0**.

Die Filter sollen explizit in verschiedenen Filterketten wiederverwendbar sein. Das führt zu bester **Modularität**: **+3**

Obwohl nicht explizit gefordert, macht es doch wenig Sinn, einen Filter zu programmieren, der die gesamte Aufnahme von der Kommunikation mit der Videoquelle über die Codierung von Bild und Ton bis zur Ausgabe in eine Datei kapselt. Daher wird dieser Stil als schwach **Separation of Concerns**-fördernd eingestuft: **+1**

Die einfache Neuanordnung der Filter, um andere Aufgaben erfüllen zu können, entspricht einer erstklassigen **Variabilität**: **+3**

Tabelle 6: Bewertung von Pipes and Filters zeigt eine übersichtliche Zusammenfassung.

| Prinzipien und Attribute | Bewertung | Qualitäten | Bewertung |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | +2 | Änderbarkeit | 0 |
| Komplexität | -2 | Erweiterbarkeit | 0 |
| Kopplung | 0 | Portability | 0 |
| Modularität | +3 | Testbarkeit | 0 |
| Separation of Concerns | +1 | Variability | +3 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 6: Bewertung von Pipes and Filters*

## *4.3 Architekturmuster*

### 4.3.1 Benutzerschnittstelle

#### *Model-View-Controller (kurz MVC)*

Das Model (in Abbildung 19: Beispiel für Model-View-Controller Kundenverwaltung genannt) besteht aus der Speicherung der Mitbestelleradressdaten und einigen Funktionen, welche die Adressdaten manipulieren können. Die View mit der Kundenliste stellt die Adressdaten in einer Tabelle dar. Wird eine Mitbestelleradresse angeklickt, so öffnet der Controller der Tabelle ein weiteres View, in dem man die Adressdaten dieses Mitbestellers ändern kann. Jede Änderung wird durch den Controller der Änderungsview überprüft. Werden die Änderungen bestätigt, so prüft der Controller die Daten abschließend und übermittelt sie anschließend an das Model. Das Model informiert nun die View mit der Übersichtstabelle von den Änderungen, damit diese die Änderungen anzeigen kann. Das Löschen von Kunden kann direkt in der Kundenliste erfolgen. Werden neue Mitbesteller angelegt, so wird ein leerer Datensatz erzeugt und die View Adressänderung zur Eingabe der Daten geöffnet.

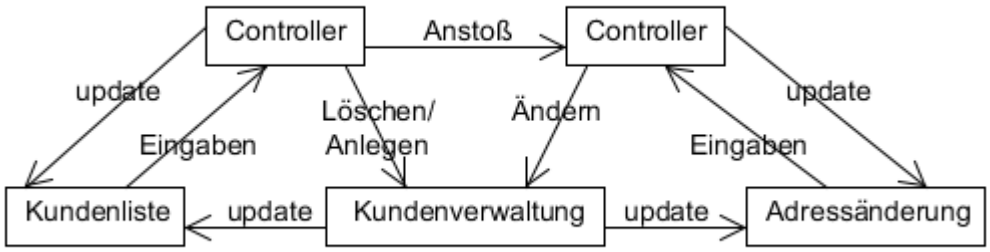


*Abbildung 19: Beispiel für Model-View-Controller*

**Change Impact** und **Kapselung** werden durch MVC nicht beeinflusst: **0**

Dieses Muster fügt 3 neue, dedizierte Komponenten zum Entwurf hinzu, die jeweils alle ihrer Aufgabe entsprechenden Funktionen beinhalten. Dadurch erhöht sich die Anzahl der Komponenten und Abhängigkeiten und damit auch die **Komplexität**: **+2**

Eine Entkopplung wird sinnigerweise durch den Einsatz weiterer Muster erreicht, ist aber nicht Bestandteil von Model-View-Controller. **Kopplung**: **0**

MVC definiert explizit 3 verschiedene Komponenten, wobei mehrere Views ausdrücklich erlaubt sind. Damit ist die **Modularität** direkt vorgeschrieben und kann nur mit einer **+3** bewertet werden.

Ebenso wie die Modularität ist die **Separation of Concerns** explizit vorgeschrieben: **+3**

Tabelle 7: Bewertung von Model-View-Controller zeigt eine übersichtliche Zusammenfassung.

| Prinzipien und Attribute | Bewertung | Qualitäten | Bewertung |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | 0 | Änderbarkeit | 0 |
| Komplexität | +2 | Erweiterbarkeit | 0 |
| Kopplung | 0 | Portability | 0 |
| Modularität | +3 | Testbarkeit | 0 |
| Separation of Concerns | +3 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 7: Bewertung von Model-View-Controller*

### *Presentation-Abstraction-Control (kurz PAC)*

Die Sammelbestellersoftware lässt sich leicht in mehrere funktionale Einheiten unterteilen. In diesem Beispiel sollen die Adress- und die Bestellverwaltung näher betrachtet werden. Jede dieser zwei Funktionseinheiten hat ein Benutzerinterface (Presentation) und eine Programmlogik (Abstraction) sowie eine eigene Komponente zur Kommunikation mit anderen Funktionseinheiten (Control). Über dieses Control kann die Bestellverwaltung z.B. Adressdaten eines Mitbestellers von der Adressverwaltung anfordern, falls die Lieferung direkt zum Mitbesteller erfolgen soll.

PAC ist somit MVC relativ ähnlich, wodurch auch die Bewertung in einigen Punkten gleich ausfällt. Daher sollen hier nur die Differenzen aufgeführt werden.

Da PAC im Vergleich zu MVC noch mehr Komponenten definiert, führt das zu einer entsprechend noch stärker ausgeprägten **Komplexität**: **+3**

Im Gegensatz zu MVC wirkt sich PAC durch die dedizierte Kommunikationskomponente entkoppelnd zwischen den Funktionseinheiten aus. Dadurch kann die **Kopplung** mit **-2** bewertet werden.

Tabelle 8: Bewertung von Presentation-Abstraction-Control zeigt eine übersichtliche Zusammenfassung.

| Prinzipien und Attribute | Bewertung | Qualitäten | Bewertung |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | 0 | Änderbarkeit | 0 |
| Komplexität | +3 | Erweiterbarkeit | 0 |
| Kopplung | -2 | Portability | 0 |
| Modularität | +3 | Testbarkeit | 0 |
| Separation of Concerns | +3 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 8: Bewertung von Presentation-Abstraction-Control*

### 4.3.2 Ablaufsteuerung

#### *Event-Based, Implicit Invocation*

Dies ist ein sehr lokal anzuwendendes Muster. Dafür wird es allerdings auch entsprechend oft eingesetzt. Es ist sogar Bestandteil anderer Muster (z.B. Inversion of Control). Hier soll es, zur Entkopplung von Model und View im Beispiel von Model-View-Controller verwendet, näher betrachtet und bewertet werden. Das Model löst hierbei ein Ereignis (einen Event) aus, wenn die Adressdaten geändert wurden. Die View hat sich im Vorfeld am Model für dieses Ereignis registriert und erhält nun eine entsprechende Meldung. Dadurch kann es anschließend die neuen Daten vom Model abrufen und diese darstellen.

Dieser Mechanismus wirkt sich nur auf die Kopplung und auf den Change Impact aus, die **restlichen Prinzipien und Attribute** werden mit **0** bewertet.

Durch den Registrierungsmechanismus, muss das Model nun nicht mehr die Adresse jeder View einprogrammiert bekommen, es führt lediglich eine Liste aller zur Laufzeit registrierten Views. Das Model wird also von den Views komplett entkoppelt, was einer negativen Wirkung auf die **Kopplung** entspricht: **-3**

Durch diese Entkopplung muss nun das Model nicht mehr angepasst werden, wenn neue Views hinzugefügt werden, denn diese registrieren sich automatisch zur Laufzeit. Da das allerdings nur eine kleine Änderung ist, wird der **Change Impact** nur wenig beeinflusst: **-1**

Tabelle 9: Bewertung von Event Based, Implicit Invocation zeigt eine übersichtliche Zusammenfassung.

| Prinzipien und Attribute | Bewertung | Qualitäten | Bewertung |
|---|---|---|---|
| Change Impact | -1 | Analysierbarkeit | 0 |
| Kapselung | 0 | Änderbarkeit | 0 |
| Komplexität | 0 | Erweiterbarkeit | 0 |
| Kopplung | -3 | Portability | 0 |
| Modularität | 0 | Testbarkeit | 0 |
| Separation of Concerns | 0 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 9: Bewertung von Event Based, Implicit Invocation*

### *Nebenläufigkeit*

Dieses Muster ist immer dann angebracht, wenn mehrere rechenintensive Aufgaben parallel ausgeführt werden können und die Vorteile eines Mehrprozessorsystems (oder einer Multicore-CPU) ausgenutzt werden sollen. Das ist zum Beispiel bei der Aufnahme eines Videos der Fall. Um Nebenläufigkeit umzusetzen, eignet sich hervorragend die Struktur, die das Pipes and Filters Muster vorgibt. Hierbei werden typischerweise die einzelnen Filter in eigenen Threads ausgeführt. Bei der Weitergabe der Streamdaten an den nächsten Filter, der in einem anderen Thread sitzt, operieren die Pipes als Mittler, so dass die Filterthreads für die Datenübergabe nicht aufeinander warten müssen. Auf diese Weise kann auf einem Prozessorcore die Rauschunterdrückung arbeiten, während ein anderer Prozessorcore die Codierung des Datenstromes vornimmt.

Die Nebenläufigkeit selbst hat somit auf die **meisten Prinzipien und Attribute** keinen Einfluss (Bewertung: **0**). Lediglich die Kapselung ist eine Voraussetzung für die parallele Ausführung mehrerer Prozesse. Würde eine Schnittstelle umgangen und auf eine Komponente zugegriffen, die gerade auf einem anderen Prozessorcore ausgeführt wird, dann kommt es mit hoher Wahrscheinlichkeit zu Zugriffsfehlern. Diese sind zudem sehr schwer zu lokalisieren und zu debuggen, weil sie meist "zufällig" auftreten. Da eine gute **Kapselung** also eine Voraussetzung für den fehlerfreien Betrieb ist, muss sie für den

Einsatz dieses Musters entsprechend umgesetzt werden. Daher kann sie als implizit durch dieses Muster beeinflusst gewertet werden und wird mit **+2** bewertet.

Durch nebenläufige Prozesse entstehen leicht Fehler, die zufällig erscheinen. Ihr Auftreten ist oft von einer zeitlichen Abhängigkeit zweier Prozesse geprägt, die nicht bei jeder Ausführung gleich ist und nicht auf jedem Prozessor auftreten muss. Aufgrund dieser scheinbaren Zufälligkeit sind Tests auf solche Fehler sehr schwierig, wenn überhaupt, durchführbar. Da alle anderen Fehler meist ganz normal testbar sind, wird die **Testbarkeit** mit **-1** bewertet.

Tabelle 10: Bewertung von Nebenläufigkeit zeigt eine übersichtliche Zusammenfassung.

| **Prinzipien und Attribute** | **Bewertung** | **Qualitäten** | **Bewertung** |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | +2 | Änderbarkeit | 0 |
| Komplexität | 0 | Erweiterbarkeit | 0 |
| Kopplung | 0 | Portability | 0 |
| Modularität | 0 | Testbarkeit | -1 |
| Separation of Concerns | 0 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 10: Bewertung von Nebenläufigkeit*

### *Inversion of Control*

Bei der Benutzung des Multimediaframeworks zur Aufnahme des analogen Videos wird in einer Initialisierungsphase das Framework nach den eigenen Wünschen eingerichtet. Das heißt, dass für die Videoquelle, der Audio- sowie auch der Videocodec, der Speicherort und die Nachbearbeitung alle Parameter entsprechend gesetzt werden. Zusätzlich werden Event-Handler eingerichtet, die während der Aufnahme aktuelle Informationen in einem Statusfenster ausgeben. Mit dem Starten der Aufnahme wird die Ablaufkontrolle an das Framework übergeben. Das Programm selbst wird jetzt nur noch über die Event-Handler eingebunden, bis die Aufnahme beendet wird.

Inversion of Control hat keinen Einfluss auf den **Change Impact**: **0**

Ein Framework stellt einen in vielen Programmen wiederwendbaren Rahmen dar, der jeweils programmspezifisch erweitert wird. Auch wenn das Framework erweitert wird, sollte es durch das Programm weiterhin genutzt werden können. Damit das funktionieren kann, muss an der Schnittstelle zwischen Programm und Framework eine gute **Kapselung** existieren: **+2**

Durch den Einsatz von Frameworks wird ein Teil der Kernfunktionalität des Programms an eine übergeordnete Instanz (das Framework) ausgelagert. Das Programm wird nunmehr lediglich noch über Events eingebunden und benötigt im Idealfall keine weiteren internen Abhängigkeiten mehr, abgesehen von der Initialisierungs- und Finalisierungsphase. Durch die verringerten Abhängigkeiten verringert sich auch die **Komplexität** des Programms: **-2**

Da das Framework bei Benutzung erweitert werden können muss, müssen entsprechende Registrierungsmechanismen für Event-Handler etc. eingebaut sein. Hier eignet sich Implicit Invocation hervorragend. Durch diese Mechanismen wird die **Kopplung** entsprechend verringert: **-2**

Durch die Einbindung des Programmcodes über viele einzelne Events werden auch die **Modularität** und die **Separation of Concerns** gefördert: **+2**

Frameworks sind speziell darauf ausgelegt, als Grundlage für eine bestimmte Art von Anwendungen benutzt und speziell für diese angepasst zu werden. Programme, die intern auf Frameworks basieren, haben daher von Haus aus eine gute **Erweiterbarkeit** (Bewertung: **+2**). Ebenso kann die **Variabilität** des Frameworks leicht auch für die Anwendung selbst verfügbar gemacht werden: **+1**

Tabelle 11: Bewertung von Inversion of Control zeigt eine übersichtliche Zusammenfassung.

| Prinzipien und Attribute | Bewertung | Qualitäten | Bewertung |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | +2 | Änderbarkeit | 0 |
| Komplexität | -2 | Erweiterbarkeit | +2 |
| Kopplung | -2 | Portability | 0 |
| Modularität | +2 | Testbarkeit | 0 |
| Separation of Concerns | +2 | Variability | +1 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 11: Bewertung von Inversion of Control*

### 4.3.3 Schnittstellen

#### *Fassade*

Multimediaframeworks haben typischerweise Schnittstellen, die nahezu jede Einstellmöglichkeit bieten, die man sich vorstellen kann. Es ist zum Beispiel möglich, die Filterkette für die Aufnahme des Videos komplett manuell aufzubauen und dabei auch jeden Filter einzeln entsprechend einzustellen. Da manche Benutzer der Frameworks sich gar nicht so genau damit beschäftigen wollen, gibt es oft auch eine Fassade mit vereinfachten Schnittstellen, über die man das Framework beispielsweise anweisen kann eine Datei abzuspielen oder einen Film von einer analogen Quelle aufzunehmen. Für die Wiedergabe wird lediglich der Pfad zu der Datei benötigt, für die Aufnahme zusätzlich der zu verwendende Codec und das Aufnahmegerät. Durch die Benutzung der Fassade braucht sich der Benutzer des Frameworks nun keine Gedanken mehr darüber zu machen, welche Filter in der Filterkette benötigt werden und wie sie verbunden werden müssen. Eine Fassade ist dabei eine Komponente des Frameworks, die ähnlich wie ein Adapter funktioniert und bei der "Übersetzung" auf die komplexeren Interfaces auf Standardeinstellungen zurückgreift.

Die Fassade fügt ein weiteres Element zur Gesamtstruktur einer Anwendung hinzu. Gleichzeitig verringert sie allerdings die Anzahl der sonst notwendigen Abhängigkeiten zum Multimediaframework, so dass sie dennoch eine verringernde Wirkung auf die **Komplexität** hat: **-2**

Durch die Benutzung der Fassade wird der Nutzer des Interfaces vom Anbieter des komplexeren Interfaces entkoppelt. Die **Kopplung** wird also verringert: **-2**

**Alle weiteren Prinzipien und Attribute** werden nicht beeinflusst: **0**

Eine Fassade wird speziell dafür eingesetzt die Verständlichkeit einer Schnittstelle zu verbessern, indem sie oft benutzte Folgen von Funktionsaufrufen in einen einzigen Funktionsaufruf zusammenfasst. Die Bewertung der **Verständlichkeit** kann daher nur **+3** sein.

Tabelle 12: Bewertung von Fassade zeigt eine übersichtliche Zusammenfassung.

| **Prinzipien und Attribute** | **Bewertung** | **Qualitäten** | **Bewertung** |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | 0 | Änderbarkeit | 0 |
| Komplexität | -2 | Erweiterbarkeit | 0 |
| Kopplung | -2 | Portability | 0 |
| Modularität | 0 | Testbarkeit | 0 |
| Separation of Concerns | 0 | Variability | 0 |
| | | Verständlichkeit | +3 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 12: Bewertung von Fassade*

### *Adapter*

Das Sammelbestellerprogramm soll die Bestellungen an einen Server des Versandhauses übertragen können. Damit nicht durch eine fehlerhaft implementierte Clientschnittstelle die Funktion des Servers beiinträchtigt wird, stellt das Versandhaus eine Clientkomponente, die nur die Kommunikation zum Server übernimmt, zur Verfügung. Das Sammelbestellerprogramm muss nun nur noch mit dieser Clientkomponente kommunizieren. Ist nun die Schnittstelle dieser Clientkomponente nicht mit der der Sammelbestellersoftware kompatibel, so kann ein Adapter, der auf der einen Seite eine zur Clientkomponente und auf der anderen eine zur Sammelbestellersoftware kompatible Schnittstelle hat, diese beiden Komponenten verbinden. Adapterintern werden die Bestelldaten entsprechend der Schnittstellenanforderungen konvertiert. Somit kann die

Sammelbestellersoftware über den Adapter und die Clientkomponente mit dem Versandhausserver kommunizieren.

Auf den **Change Impact** hat der Adapter keine Auswirkungen: **0**

Ändert das Versandhaus jemals seine Clientkomponente, so muss nun nicht mehr das ganze Sammelbestellerprogramm angepasst, sondern lediglich der Adapter entsprechend aktualisiert werden. Somit bewirkt der Adapter einen Rückgang in der **Kopplung** der beiden Komponenten, zwischen denen er eingesetzt ist. Bewertung: **-2**

Damit diese Anpassung problemlos möglich ist, müssen die Komponenten beidseitig des Adapters gut gekapselt sein, es dürfen keine über die Schnittstellendefinition hinausgehenden Annahmen bei der Benutzung derselben gemacht werden, da der Adapter nur zwischen den Schnittstellenspezifikationen adaptieren kann. Folglich bewirkt der Einsatz eines Adapters eine Verbesserung der **Kapselung**: **+2**

Mit dem Adapter erhöht sich die Anzahl der Komponenten des Entwurfes, was zu einem leichten Anstieg der **Komplexität** führt: **+1**

Auf die **Modularität** und die **Separation of Concerns** hat ein Adapter keinen Einfluss: **0**

Der Adapter passt inkompatible Schnittstellen aneinander an und ermöglicht so die Wiederverwendung ansonsten inkompatibler Komponenten. **Wiederverwendbarkeit**: **+3**

Tabelle 13: Bewertung von Adapter zeigt eine übersichtliche Zusammenfassung.

| **Prinzipien und Attribute** | **Bewertung** | **Qualitäten** | **Bewertung** |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | +2 | Änderbarkeit | 0 |
| Komplexität | +1 | Erweiterbarkeit | 0 |
| Kopplung | -2 | Portability | 0 |
| Modularität | 0 | Testbarkeit | 0 |
| Separation of Concerns | 0 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | +3 |

*Tabelle 13: Bewertung von Adapter*

### *Dependency Injection*

In einem Multimediaframework liegen die Interfaces zwischen Framework und Filter in Form von Headerdateien vor. An diese Interfaces müssen sich sowohl der Programmierer des Filters als auch der des Frameworks halten. Zudem muss der Filter das Interface komplett implementieren, da es bei einem Aufruf einer nicht implementierten Methode ansonsten zu einer Access Violation kommt. Bevor das Framework den Filter tatsächlich benutzen kann, müssen zusätzlich zu den Interfaces auch die Einsprungadressen der Methoden bekannt sein. Dazu reicht es aus, die Startadresse des das Interface implementierenden Objektes im Speicher zu übergeben. Da die innere Struktur des Objektes durch die Interfaces bekannt ist, können nun auch die Einsprungadressen der Methoden bestimmt und das Objekt benutzt werden. Der Vorgang der Übergabe der Startadresse ist der Kern von Dependency Injection.

Dieses Muster setzt den **Change Impact** herunter, da nun auf gleiche Weise verschiedene Filter registriert werden können, ohne den Quelltext des Frameworks verändern zu müssen: **-2**

Durch die explizite Verwendung von beidseitig bekannten Interfaces (identische Headerdateien) und die spätere Injektion der eigentlichen Implementierung, ist die **Kapselung** im höchsten Maße gegeben: **+3**

Die **Komplexität** des Entwurfs wird durch Dependency Injection nicht beeinflusst: **0**

Durch die Injektion des Zeigers auf das implementierende Objekt zur Laufzeit des Programms, ist das Framework vom Plugin entkoppelt. **Kopplung**: **-2**

**Modularität** und **Separation of Concerns** werden durch Dependency Injection nicht beeinflusst: **0**

Tabelle 14: Bewertung von Dependency Injection zeigt eine übersichtliche Zusammenfassung.

| Prinzipien und Attribute | Bewertung | Qualitäten | Bewertung |
|---|---|---|---|
| Change Impact | -2 | Analysierbarkeit | 0 |
| Kapselung | +3 | Änderbarkeit | 0 |
| Komplexität | 0 | Erweiterbarkeit | 0 |
| Kopplung | -2 | Portability | 0 |
| Modularität | 0 | Testbarkeit | 0 |
| Separation of Concerns | 0 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 14: Bewertung von Dependency Injection*

### 4.3.4 Verteilung

#### *Broker*

Das Sammelbestellerprogramm soll als verteiltes System entworfen werden. Dabei sollen dedizierte Server für die Datenhaltung, den Druck und die Kommunikation mit dem Versandhaus (Kommunikation aus dem LAN ins WAN) zum Einsatz kommen, die leicht austausch- und relokierbar sein sollen. Dazu wird auf jedem Client ein Broker eingesetzt. Dieser sendet bei Programmstart eine Broadcast-Message ins Netzwerk. Anschließend melden sich alle verfügbaren Services beim Broker an und registrieren dabei ihre Services. Das Sammelbestellerprogramm nutzt allein den Broker zum Zugriff auf die Services und braucht sich so nicht um die Lokalisierung und die Kommunikation zu den einzelnen Servern zu kümmern.

Durch den Registrierungsalgorithmus wird der **Change Impact** herabgesetzt: **-1**

Dieses Muster ist explizit dafür ausgelegt, Komponenten einfach austauschen und relokieren zu können. Dazu zählen Server und Broker gleichermaßen. Damit das Zusammenspiel dennoch funktioniert, müssen alle Komponenten sich strikt an die Schnittstellenspezifikation halten. **Kapselung**: **+2**

Die **Komplexität** des Systems wird durch die zusätzliche Brokerkomponente erhöht: **+2**

Die **Kopplung** wird durch den Registrierungsmechanismus stark herabgesetzt: **-3**

Das Muster motiviert die strukturelle sowie auch inhaltliche Trennung der einzelnen Dienste. Die **Separation of Concerns** wird somit leicht unterstützt (Bewertung: **+1**), die **Modularität** wird zusätzlich durch die sehr leichte Austauschbarkeit der Server unterstützt: **+2**

Durch den Einsatz eines Brokers werden Änderungen an den verfügbaren Services und deren Relokation vereinfacht. Das hat direkte Auswirkungen auf die **Änderbarkeit** der Software: **+1**

Tabelle 15: Bewertung von Broker zeigt eine übersichtliche Zusammenfassung.

| **Prinzipien und Attribute** | **Bewertung** | **Qualitäten** | **Bewertung** |
|---|---|---|---|
| Change Impact | -1 | Analysierbarkeit | 0 |
| Kapselung | +2 | Änderbarkeit | +1 |
| Komplexität | +2 | Erweiterbarkeit | 0 |
| Kopplung | -3 | Portability | 0 |
| Modularität | +2 | Testbarkeit | 0 |
| Separation of Concerns | +1 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 15: Bewertung von Broker*

### *Proxies*

Wenn das Sammelbestellerprogramm erfolgreich ist, wird der Server beim Versandhaus irgendwann an seine Leistungsgrenzen stoßen. Dann kann ein Proxy zum Einsatz kommen, der die eingehenden Anfragen von den Sammelbestellern auf syntaktische Gültigkeit überprüft (und somit z.B. Spam und andere Angriffe filtert) und einen Cache hält. Dieser Cache kann zum Beispiel die Lieferzeit von oft bestellten Artikeln puffern und somit den eigentlichen Server entlasten.

Der Proxy hat keinen Einfluss auf den **Change Impact**: **0**

Da der Proxy lediglich den Kommunikationskanal zwischen Client und Server beeinflusst, kann er sich bei der Umsetzung seiner Aufgaben auch nur auf diese Schnittstelle berufen.

Client und Server sind meist andere Programme und können jederzeit ausgetauscht werden, ohne dass der Proxy davon erfahren muss. Daher ist eine gute **Kapselung** Voraussetzung für den störungsfreien Betrieb des Proxies: **+2**

Das hinzufügen eines Proxies in ein System führt zu einer Erhöhung der Anzahl der Komponenten und somit zu einer Erhöhung der **Komplexität**: **+1**

Da sich der Proxy im Kommunikationskanal zwischen Client und Server befindet, verringert sich die **Kopplung** entsprechend: **-2**

Da der Proxy vollkommen transparent agiert, kann er jederzeit entfernt oder ersetzt werden. **Modularität**: **+2**

Da das Muster genau vorschreibt was der Proxy zu tun hat, wirkt es sich positiv auf die **Separation of Concerns** aus: **+2**

Tabelle 16: Bewertung von Proxies zeigt eine übersichtliche Zusammenfassung.

| **Prinzipien und Attribute** | **Bewertung** | **Qualitäten** | **Bewertung** |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | +2 | Änderbarkeit | 0 |
| Komplexität | +1 | Erweiterbarkeit | 0 |
| Kopplung | -2 | Portability | 0 |
| Modularität | +2 | Testbarkeit | 0 |
| Separation of Concerns | +2 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 16: Bewertung von Proxies*

### 4.3.5 Adaptierbarkeit

***Mikrokernel***

Das Sammelbestellerprogramm soll unabhängig von Betriebssystem und Benutzerinterface implementiert werden. Eine mögliche Mikrokernelarchitektur kann wie in Abbildung 20: Beispiel für Mikrokernel aussehen.

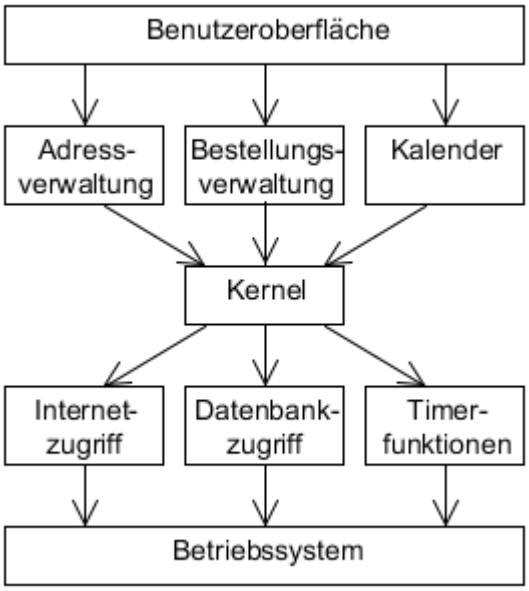


*Abbildung 20: Beispiel für Mikrokernel*

Der Mikrokernel selbst stellt grundlegende Funktionalität, wie beispielsweise Datei-, Internet- und Datenbankzugriff, Verwaltung von Listen und Tabellen und Timerfunktionen zur Verfügung. Interne Server realisieren diese für das jeweilig verwendete Betriebssystem. Die externen Server realisieren damit die Adressverwaltung der Mitbesteller, das Handling der Bestellungen und eine Kalenderfunktion für die Verwaltung von Verkaufsaktionen. Ein Adapter für ein lokales Benutzerinterface kann wahrscheinlich eingespart werden. Für ein mögliches Webinterface könnte ein Adapter benötigt werden, der die Interfaces der externen Server für die serverseitige Programmiersprache leicht verwendbar macht.

Der **Change Impact** wird durch dieses Muster nicht beeinflusst: **0**

Laut [BMRSS96] dient das Muster vor allem einer guten Anpassbarkeit an neue Umgebungen und Anforderungen, die über den Austausch interner bzw. externer Server realisiert wird. Da dies ohne eine gute **Kapselung** nicht reibungslos durchführbar ist, kann sie als implizit vorgeschrieben angesehen werden: **+2**

Als neue Komponenten werden der Kernel, die internen Server und ggf. der Adapter als neue Komponenten eingeführt. Dabei haben alle genau eine Abhängigkeit zum Mikrokernel. Die Anwendungen (Benutzeroberflächen des Sammelbestellerprogramms) können Abhängigkeiten zu mehreren externen Servern haben. Somit ergibt sich aus der Sterntopologie im Bereich interne Server – Kernel – externe Server eine minimale Komplexität. Diese wird allerdings durch das Hinzufügen neuer Komponenten durch das Muster und die potentielle n-zu-n-Topologie zwischen externen Servern und Anwendungen wieder erhöht. Insgesamt wird die **Komplexität** mit **-1** bewertet.

Der Kernel wird durch die internen Server von der Hardware bzw. dem Betriebssystem entkoppelt. Die Anwendungen können durch einen Adapter von den externen Servern entkoppelt werden. Somit ergibt sich für die **Kopplung** eine Bewertung von **-2**.

Die **Modularität** wird durch die Unterteilung in verschiedene externe Server, den Kernel und die internen Server stark unterstützt: **+3**

Zum Teil wird die Separation of Concerns direkt durch das Muster vorgeschrieben (interne Server bilden die Abstraktion von Hardware bzw. Betriebssystem). Zudem wird sie durch die Verwendung verschiedener externer Server nahegelegt. Aus diesen Gründen wird die **Separation of Concerns** mit **+2** bewertet.

Die **Erweiterbarkeit** ist durch einfaches Hinzufügen weiterer externer Server gegeben: **+3**

Die Verwendung interner Server und gegebenenfalls Adapter zwischen externen Servern und Benutzeroberfläche wirkt sich positiv auf die **Portability** aus: **+2**

Tabelle 17: Bewertung von Mikrokernel zeigt eine übersichtliche Zusammenfassung.

| Prinzipien und Attribute | Bewertung | Qualitäten | Bewertung |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | +2 | Änderbarkeit | 0 |
| Komplexität | -1 | Erweiterbarkeit | +3 |
| Kopplung | -2 | Portability | +2 |
| Modularität | +3 | Testbarkeit | 0 |
| Separation of Concerns | +2 | Variability | 0 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 17: Bewertung von Mikrokernel*

### *Reflection*

Wenn die Sammelbestellersoftware für mehrere Versandhäuser ausgelegt werden soll, dann muss sie für die verschiedenen Versandhäuser auch verschiedene Werte für eine Bestellung speichern können. Das Handling der unterschiedlichen Bestellungstypen kann entweder zur Designtime koordiniert werden oder mittels Reflection zur Laufzeit. Da dieses Handling zu keinem zeitkritischen Punkt ansteht und die Laufzeitvariante leichter umzusetzen ist, ist letztere in diesem Fall vorzuziehen. Um die Laufzeiteffizienz möglichst wenig zu beeinflussen, wird eine Klasse Bestellung definiert, die alle Werte enthält, mit denen das Programm später noch Berechnungen anstellen muss und die für jedes Versandhaus gleich sind. Dazu gehören z.B. der Preis, der Mitbesteller, der den Artikel bestellt hat und die Anzahl der Raten. Die Regeln für die restlichen, versandhausabhängigen Werte, wie etwa Bestellnummer, Farbnummer und Größe, werden in einem Dialog für jedes Versandhaus einzeln definiert (siehe Abbildung 21: Dialog zum Beispiel für Reflection). Anhand dieser Definitionen wird dynamisch zur Laufzeit eine versandhausspezifische Klasse von der Klasse Bestellung abgeleitet (siehe Abbildung 22: Klassendiagramm zum Beispiel für Reflection), welche die zusätzlichen Werte nach den Regeln aus dem Dialog enthält. Die abgeleitete Klasse kann bei der Berechnung von Ratenplänen als Klasse Bestellung behandelt werden, lediglich bei der Speicherung, Darstellung, Bearbeitung und Übertragung zum Versandhaus müssen die zusätzlichen Werte mittels Reflection interpretiert werden. Diese Interpretation kann nur mittels Laufzeitinformationen erfolgen, da die Klasse erst zur Laufzeit erstellt wurde.

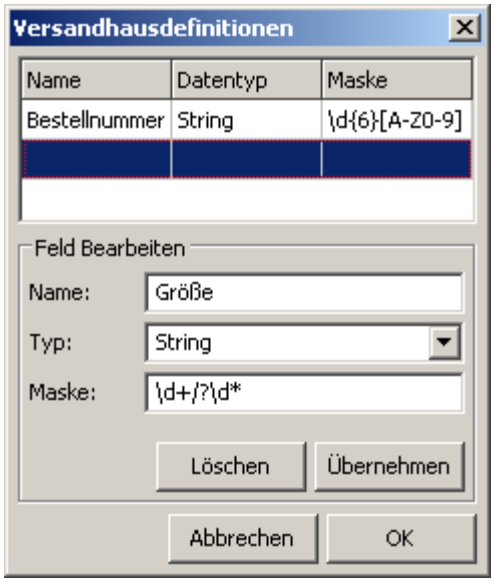


*Abbildung 21: Dialog zum Beispiel für Reflection*

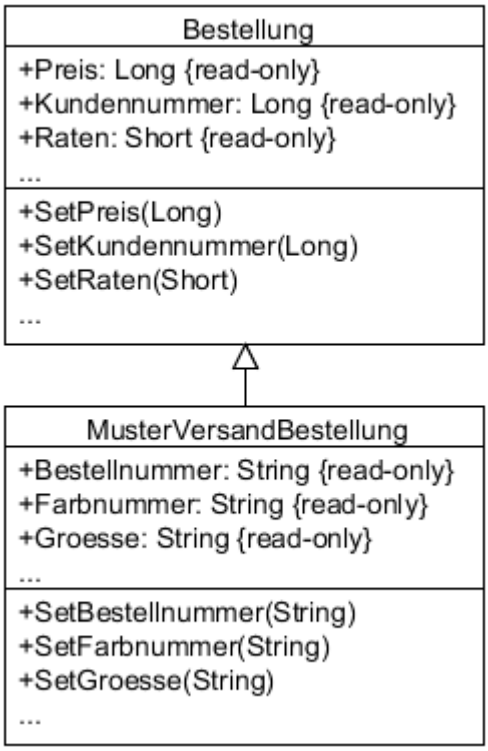


*Abbildung 22: Klassendiagramm zum Beispiel für Reflection*

Bei Änderungen an einem versandhausspezifischen Feld muss lediglich die Definition dieses Feldes im Dialog angepasst werden, weitere Änderungen sind durch die dynamische Interpretation der Felder nicht vonnöten. Somit wird der **Change Impact** durch Reflection erheblich verringert: **-3**

Die dynamische Erzeugung von Klassen zur Laufzeit erfordert eine Vorhaltung der dazu notwendigen Informationen. Diese müssen laut Musterbeschreibung in eigenen Komponenten gehalten werden, so dass dadurch die **Komplexität** steigt: **+2** Ebenso wird dadurch allerdings auch die **Modularität** und **Separation of Concerns** unterstützt, was mit **+1** und **+2** bewertet wird.

Der große Vorteil von Reflection ist die leichte **Änderbarkeit** aller Teile der Software, die reflexiv programmiert wurden, selbst Strukturänderungen lassen sich direkt über die Benutzeroberfläche realisieren: **+3**

Manche Teile von Reflection benötigen die Unterstützung der Laufzeitumgebung, was die **Portability** leicht einschränkt: **-1**

Durch die vielen Freiheitsgrade, die durch reflexive Programmierung entstehen, ist es enorm schwierig bis unmöglich alle Konstellationen vorauszusehen und zu testen, was zu einer Bewertung der **Testbarkeit** von **-2** führt. Gleichzeitig wird dadurch allerdings die **Variabilität** der Software erhöht: **+2**

Durch die Trennung von Strukturinformationen und Anwendungslogik wird die **Verständlichkeit** eingeschränkt: **-2**

Tabelle 18: Bewertung von Reflection zeigt eine übersichtliche Zusammenfassung.

| **Prinzipien und Attribute** | **Bewertung** | **Qualitäten** | **Bewertung** |
|---|---|---|---|
| Change Impact | -3 | Analysierbarkeit | 0 |
| Kapselung | 0 | Änderbarkeit | +3 |
| Komplexität | +2 | Erweiterbarkeit | 0 |
| Kopplung | 0 | Portability | -1 |
| Modularität | +1 | Testbarkeit | -2 |
| Separation of Concerns | +2 | Variability | +2 |
| | | Verständlichkeit | -2 |
| | | Wiederverwendbarkeit | 0 |

*Tabelle 18: Bewertung von Reflection*

### *Plug-In*

In einem Framework für die Bearbeitung von Multimedia, wie es beispielsweise bei der Aufnahme eines Videos benutzt wird, sind die Filter im typischen Fall Plugins. Auf diese Weise ist es leicht, neue Codecs ins System zu integrieren. Die Plugin-Schnittstelle muss dann eine Obermenge der Filterein- und -ausgabeschnittstellen sein.

Dieses Muster wirkt sich nur auf Kapselung und Modularität aus, **alle anderen Prinzipien und Attribute** werden daher mit **0** bewertet.

Da bei der Fertigstellung des Multimediaframeworks die Filter-Plugins noch nicht existieren, muss sich ein Framework vollständig auf die Schnittstellen verlassen. Das erfordert implizit **Kapselung**: **+2**

Das Muster schreibt explizit eine eigene Komponente für den gewünschten Funktionsumfang vor, die erst zur Laufzeit ins Programm eingebunden wird. Das ist das höchstmögliche Maß an **Modularität**: **+3**

Mittels Plug-Ins lässt sich eine Software erweitern, ohne den Kern zu ändern. Der Benutzer der Software muss meist nur eine zusätzliche Datei im Programmverzeichnis speichern und schon kann die Software mehr. Somit sind Plugins der Inbegriff der **Erweiterbarkeit**: **+3**

Benutzt eine Software Plugins, so lässt sie sich variabel zusammenstellen, indem die entsprechenden Plugins mitgeliefert werden. **Variabilität**: **+1**

Da Plugins speziell für eine Software geschriebene Anwendungsteile sind, lassen sie sich nur schwerlich in anderen Anwendungen wiederverwenden. **Wiederverwendbarkeit**: **-1**

Tabelle 19: Bewertung von Plug-In zeigt eine übersichtliche Zusammenfassung.

| **Prinzipien und Attribute** | **Bewertung** | **Qualitäten** | **Bewertung** |
|---|---|---|---|
| Change Impact | 0 | Analysierbarkeit | 0 |
| Kapselung | +2 | Änderbarkeit | 0 |
| Komplexität | 0 | Erweiterbarkeit | +3 |
| Kopplung | 0 | Portability | 0 |
| Modularität | +3 | Testbarkeit | 0 |
| Separation of Concerns | 0 | Variability | +1 |
| | | Verständlichkeit | 0 |
| | | Wiederverwendbarkeit | -1 |

*Tabelle 19: Bewertung von Plug-In*

# 5 Auswertung und Klassifikation

## *5.1 Auswertung der Bewertungen*

Die in Kapitel 4 - Bewertung der Muster gefundenen direkten Bewertungen der einzelnen Kriterien werden nun mittels der in Kapitel 3.1.2 - Hierarchie der Bewertungskriterien beschriebenen Hierarchie zu einer Gesamtbewertung der einzelnen Qualitäten verrechnet. Zu diesem Zweck hat der Autor mit der Freeware OpenOffice.org Calc 3.1.1 ein Tabellendokument entworfen, welches die notwendigen Berechnungen ausführt. Dieses ist auf der beigelegten CD unter dem Dateinamen Berechnungsvorlage.ods zu finden.

### 5.1.1 Beschreibung der Berechnungsvorlage

Dieses Dokument besteht aus insgesamt 10 Tabellen, in denen in 4 Schritten die Endwerte errechnet werden. In die Tabelle "direkte Einflüsse" werden die in Kapitel 4 - Bewertung der Muster gefundenen direkten Bewertungen eingetragen. Jeder Verrechnungsschritt spiegelt eine Ebene in der Hierarchie wieder (siehe Abbildung 23: Zuordnung der Ebenen der Hierarchie auf die Tabellenblätter) und ist mittels zweier Tabellen realisiert. Für jeden Schritt gibt es eine Tabelle "Abhängigkeiten", in die die Abhängigkeiten zwischen den Kriterien mit ihren entsprechenden Kantengewichten eingetragen werden und eine Tabelle "Ergebnisse", die alle bis dahin errechneten Bewertungen für den nächsten Berechnungsschritt bereitstellt. Die letzte Ergebnisseite "Ergebnisse 4" dient als Grundlage für die Tabelle "Endergebnisse", welche neben einer tabellarischen Darstellung auch ein Diagramm beinhaltet.

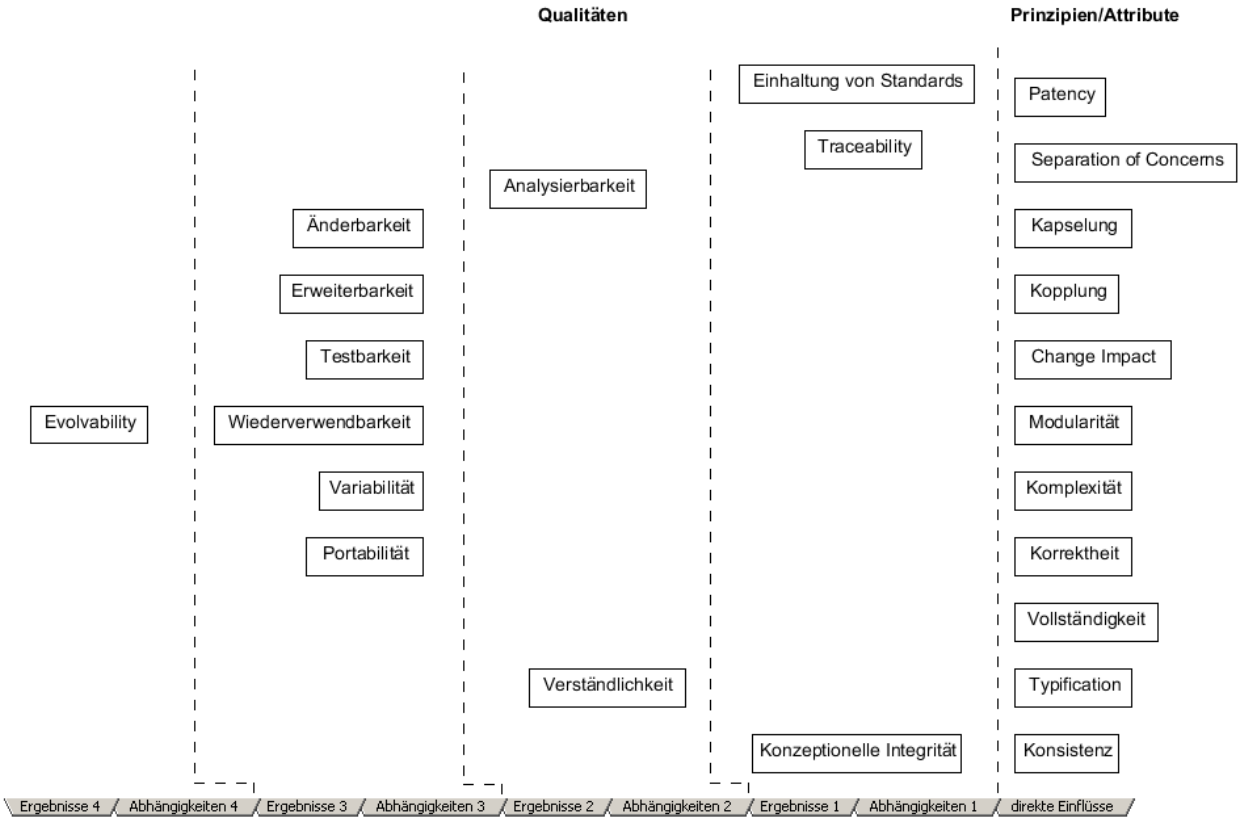


*Abbildung 23: Zuordnung der Ebenen der Hierarchie auf die Tabellenblätter*

### *Die Tabelle "Abhängigkeiten"*

In jeder "Abhängigkeiten"-Tabelle befindet sich eine n*n Matrix, die die Abhängigkeiten der Kriterien von anderen Kriterien darstellt. Betrachtet man eine Zeile, so werden dort die Abhängigkeiten des links stehenden Kriteriums eingetragen. Jede Spalte steht für eine Abhängigkeit dieses Kriteriums. Das Verhältnis der Beträge ist die Gewichtung der Abhängigkeiten, das Vorzeichen steht für positiven bzw. negativen Einfluss. Ist ein Kriterium nur von sich selbst abhängig (es steht lediglich eine 1 in der Hauptdiagonalen), so wird der Wert der vorhergehenden Ergebnistabelle bzw. der Tabelle "direkte Einflüsse" einfach übernommen. Dieser Wert repräsentiert auch den Einfluss der direkten Bewertung, wenn andere Abhängigkeiten in der Zeile existieren. Rechts von der Abhängigkeitsmatrix existiert noch eine Spalte für die Summe aller Beträge in der Zeile. Diese wird bei der Berechnung des Verhältnisses einer jeden Abhängigkeit als Teiler benötigt, um die Summe der Abhängigkeiten auf 1 zu normalisieren.

Beispielsweise steht in "Abhängigkeiten 2" in der Zeile für die Verständlichkeit und der Spalte Komplexität eine -2. Das bedeutet, dass die Verständlichkeit von der Komplexität abhängt. Das negative Vorzeichen drückt aus, dass eine hohe Komplexität tendenziell zu einer niedrigen Verständlichkeit führt. Die Summe aller Beträge in der Zeile Verständlichkeit ist 15. Das bedeutet, dass die Verständlichkeit zu 2/15 von der Komplexität abhängt. Durch den Teiler ist gewährleistet, dass die Summe aller Einflüsse immer genau 1 ist. Wäre dies nicht so, würde dies das Bewertungsintervall (-3..3) beeinflussen.

### *Die Tabelle "Ergebnisse"*

Die Tabelle "direkte Einflüsse" kann als initiale Ergebnistabelle angesehen werden. Hier sind alle direkten Einflüsse der Muster auf die Kriterien erfasst, die zusätzlich zu den durch Abhängigkeiten erfassten Einflüssen Beachtung finden müssen. In der Tabelle "Abhängigkeiten 1" sind alle Abhängigkeiten der untersten Ebene der Hierarchie, also von Einhaltung von Standards, Traceability und konzeptionelle Integrität, eingetragen. Da das relativ wenige Abhängigkeiten sind, ist die Matrix dünn besetzt und gleicht fast einer Einheitsmatrix. Zudem ist keine der Abhängigkeiten dieser drei Kriterien bewertet worden, so dass die Tabelle "Ergebnisse 1" die gleichen Daten enthält wie die Tabelle "direkte Einflüsse".

Die Bewertung der Fassade bzgl. des Kriteriums Verständlichkeit, welche hier als Beispiel dienen soll, ergibt sich aus einer Matrixmultiplikation zweier Vektoren. Der erste ist der Zeilenvektor der Zeile der Verständlichkeit aus der Abhängigkeitsmatrix in der Tabelle "Abhängigkeiten 2". Der andere ist der Zeilenvektor der Fassade auf der vorhergehenden Ergebnisseite "Ergebnisse 1". Nun wird der Zeilenvektor der Abhängigkeiten mit dem transponierten Zeilenvektor des vorherigen Ergebnisses multipliziert und das Ergebnis durch die Summe der Beträge des Abhängigkeitsvektors dividiert. Durch diese Division wird das Ergebnis wieder auf den Bereich -3..3 normalisiert. Bei der Matrixmultiplikation wird im Prinzip für jedes Kriterium das Gewicht seiner Auswirkung auf die Verständlichkeit (ggf. 0, falls keine Auswirkung) mit der Bewertung des Musters Fassade bzgl. des Kriteriums multipliziert und alle diese Produkte addiert. Nach der Division ist das Ergebnis die Bewertung des Musters Fassade bzgl. der Verständlichkeit inclusive aller Abhängigkeiten. In der Formel in der Berechnungsvorlage ist durch eine WENN-Funktion zusätzlich der Fall abgefangen, dass die Summe aller Beträge der Abhängigkeiten Null ist, um einer Division durch Null vorzubeugen.

### *Endergebnis*

Nach dem schrittweisen Aufbau der Bewertungen über die "Abhängigkeiten"- und "Ergebnisse"-Tabellen stehen in "Ergebnisse 4" alle Bewertungen incl. aller Abhängigkeiten. Hier gibt es eine weitere Zeile unter den Ergebnissen, in der nur in den Spalten eine 1 steht, die mindestens für ein Muster eine Bewertung ungleich Null beinhalten. Andernfalls kann davon ausgegangen werden, dass dieses Kriterium nicht bewertet wurde und es wird eine Null dargestellt.

Zur besseren Trennung der Ergebnistabelle von der Berechnung wurde das Tabellenblatt "Endergebnisse" eingefügt, auf welchem die Ergebnisse der Bewertung ansprechend gestaltet und den eigenen Wünschen angepasst präsentiert werden können. Der Autor wählte eine tabellarische Darstellung (Tabelle 20: Ergebnisse der Bewertung), ein an die Qualitäten angepasstes Diagramm (Abbildung 24: Ergebnisse der Bewertung) und ein Diagramm zur Darstellung der Bewertung der Evolvability (Abbildung 25: Bewertung der Evolvability). Mit "n.b." markierte Zellen in der Tabelle gehören zu nicht bewerteten Kriterien. Das Liniendiagramm wurde gewählt, weil dadurch die vielen gleichen Symbole mit teilweise ähnlichen Farben leichter zuordenbar sind, sie sollen keinen Werteverlauf suggerieren.

| Muster \ Kriterien | Change Impact | Kapselung | Komplexität | Konsistenz | Kopplung | Korrektheit | Modularität | Patency | Separation of Concerns | Typification | Vollständigkeit | Analysierbarkeit | Änderbarkeit | Einhaltung von Standards | Erweiterbarkeit | konzeptionelle Integrität | Portability | Testbarkeit | Traceability | Variability | Verständlichkeit | Wiederverwendbarkeit | Evolvability |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Client-Server | 0 | 0 | -3 | n.b. | 0 | n.b. | 3 | n.b. | 1 | n.b. | n.b. | 0,25 | 0,48 | n.b. | 1,14 | n.b. | 0,83 | 1,17 | n.b. | 0,75 | 0,67 | 0,38 | 0,51 |
| Layers | 0 | 0 | -3 | n.b. | 0 | n.b. | 3 | n.b. | 2 | n.b. | n.b. | 0,25 | 0,5 | n.b. | 0,85 | n.b. | 1,83 | 1,33 | n.b. | 0,75 | 0,73 | 0,38 | 0,61 |
| Repository | 0 | 0 | 1 | n.b. | 0 | n.b. | 1 | n.b. | 2 | n.b. | n.b. | 0,08 | 0,03 | n.b. | 0,17 | n.b. | 0,34 | 0,33 | n.b. | 0,25 | 0,07 | 0,13 | 0,12 |
| Blackboard | 0 | 0 | -2 | n.b. | -1 | n.b. | 1 | n.b. | 2 | n.b. | n.b. | 0,17 | 0,34 | n.b. | 0,42 | n.b. | 0,54 | 1 | n.b. | 0,25 | 0,53 | 0,13 | 0,29 |
| Batchprogramme | 0 | 3 | -3 | n.b. | -3 | n.b. | 3 | n.b. | 0 | n.b. | n.b. | 1,08 | 1,7 | n.b. | 1,6 | n.b. | 0,97 | 1 | n.b. | 0,75 | 1,4 | 0,75 | 0,84 |
| Pipes and Filters | 0 | 2 | -2 | n.b. | 0 | n.b. | 3 | n.b. | 1 | n.b. | n.b. | 0,42 | 0,57 | n.b. | 0,91 | n.b. | 0,89 | 0,67 | n.b. | 2,25 | 0,67 | 0,63 | 0,59 |
| Model-View-Controller | 0 | 0 | 2 | n.b. | 0 | n.b. | 3 | n.b. | 3 | n.b. | n.b. | 0,25 | 0,13 | n.b. | 0,54 | n.b. | 0,74 | 0,67 | n.b. | 0,75 | 0,13 | 0,38 | 0,31 |
| Presentation-Abstraction-Control | 0 | 0 | 3 | n.b. | -2 | n.b. | 3 | n.b. | 3 | n.b. | n.b. | 0,42 | 0,21 | n.b. | 0,61 | n.b. | 0,7 | 0,83 | n.b. | 0,75 | 0,13 | 0,38 | 0,35 |
| Implicit Invocation | -1 | 0 | 0 | n.b. | -3 | n.b. | 0 | n.b. | 0 | n.b. | n.b. | 0,25 | 0,44 | n.b. | 0,25 | n.b. | 0,03 | 0,5 | n.b. | 0 | 0,2 | 0 | 0,15 |
| Nebenläufigkeit | 0 | 2 | 0 | n.b. | 0 | n.b. | 0 | n.b. | 0 | n.b. | n.b. | 0,17 | 0,16 | n.b. | 0,13 | n.b. | 0,13 | -0,5 | n.b. | 0 | 0,13 | 0,25 | 0,06 |
| Inversion of Control | 0 | 2 | -2 | n.b. | -2 | n.b. | 2 | n.b. | 2 | n.b. | n.b. | 0,5 | 0,66 | n.b. | 1,13 | n.b. | 0,85 | 1 | n.b. | 1 | 0,8 | 0,5 | 0,57 |
| Fassade | 0 | 0 | -2 | n.b. | -2 | n.b. | 0 | n.b. | 0 | n.b. | n.b. | 0,17 | 0,43 | n.b. | 0,32 | n.b. | 0,18 | 0,67 | n.b. | 0 | 1 | 0 | 0,23 |
| Adapter | 0 | 2 | 1 | n.b. | -2 | n.b. | 0 | n.b. | 0 | n.b. | n.b. | 0,33 | 0,24 | n.b. | 0,2 | n.b. | 0,08 | -0,17 | n.b. | 0 | 0,13 | 0,63 | 0,12 |
| Dependency Injection | -2 | 3 | 0 | n.b. | -2 | n.b. | 0 | n.b. | 0 | n.b. | n.b. | 0,42 | 0,8 | n.b. | 0,43 | n.b. | 0,21 | -0,17 | n.b. | 0 | 0,33 | 0,38 | 0,24 |
| Broker | -1 | 2 | 2 | n.b. | -3 | n.b. | 2 | n.b. | 1 | n.b. | n.b. | 0,58 | 0,82 | n.b. | 0,69 | n.b. | 0,49 | 0,33 | n.b. | 0,5 | 0,27 | 0,5 | 0,39 |
| Proxies | 0 | 2 | 1 | n.b. | -2 | n.b. | 2 | n.b. | 2 | n.b. | n.b. | 0,5 | 0,43 | n.b. | 0,64 | n.b. | 0,66 | 0,5 | n.b. | 0,5 | 0,4 | 0,5 | 0,37 |
| Mikrokernel | 0 | 2 | -1 | n.b. | -2 | n.b. | 3 | n.b. | 2 | n.b. | n.b. | 0,58 | 0,66 | n.b. | 1,43 | n.b. | 1,85 | 1 | n.b. | 0,75 | 0,73 | 0,63 | 0,72 |
| Reflection | -3 | 0 | 2 | n.b. | 0 | n.b. | 1 | n.b. | 2 | n.b. | n.b. | 0,08 | 1,07 | n.b. | 0,21 | n.b. | -0,19 | -0,17 | n.b. | 1,25 | -0,47 | 0,13 | 0,19 |
| Plug-In | 0 | 2 | 0 | n.b. | 0 | n.b. | 3 | n.b. | 0 | n.b. | n.b. | 0,42 | 0,4 | n.b. | 1,23 | n.b. | 0,65 | 0,17 | n.b. | 1,25 | 0,33 | 0,5 | 0,45 |

*Tabelle 20: Ergebnisse der Bewertung*

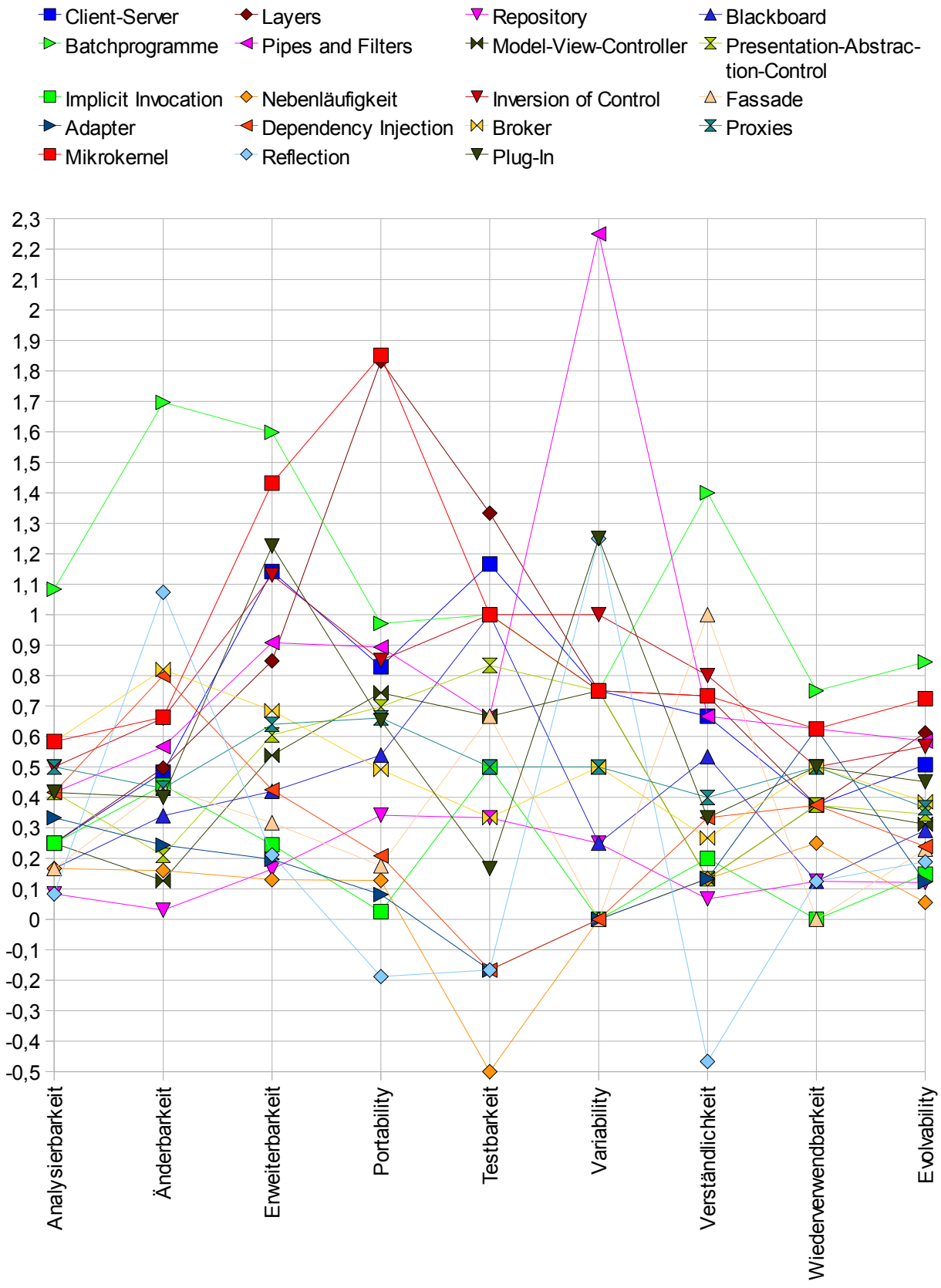


*Abbildung 24: Ergebnisse der Bewertung*

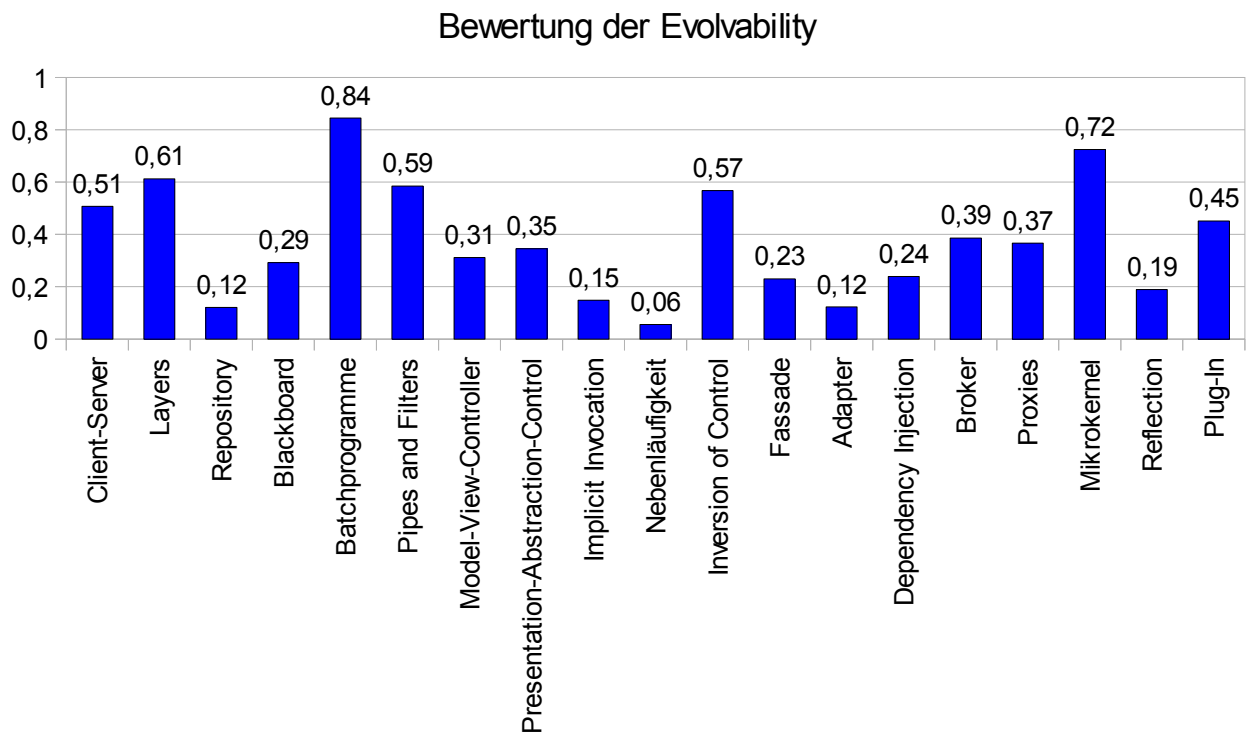


*Abbildung 25: Bewertung der Evolvability*

## 5.1.2 Interpretation der Bewertungen

### *Geringer Betrag der Bewertung*

Das Erste, was dem Betrachter der Endergebnisse auffällt, ist der kleine Betrag der Bewertungen der Muster bezüglich Evolvability. Wie schon im Kapitel 3.1.2 - Hierarchie der Bewertungskriterien beschrieben, wurde die Hierarchie dafür ausgelegt, die Auswirkungen auf eine aus den Mustern resultierende Software zu bewerten. Da eine Software aber auch Qualitäten aufweisen kann, die durch Muster nicht beeinflussbar sind (siehe Kapitel 3.3 - Bewertbarkeit der Kriterien), sich aber auf die Evolvability auswirken, kann die Bewertung der Muster den Maximalwert von +3 für Evolvability gar nicht erreichen. Eine solche Bewertung würde nämlich bedeuten, dass die Evolvability allein durch den Einsatz dieses Musters allumfassend sichergestellt werden kann.

### *Tauglichkeit der Muster für die Evolvability*

#### Batchprogramme

Batchprogramme wirken sich im Vergleich zu den anderen Mustern auffällig gut auf die bewerteten Qualitäten aus. Für die Evolvability ergibt sich somit ein Wert von **0,84**. Diesem guten Ergebnis steht allerdings die beschränkte Einsetzbarkeit dieses Musters gegenüber, es ist lediglich dazu geeignet, die aufeinanderfolgende Ausführung mehrerer Tools zu automatisieren.

#### Mikrokernel

Wie schon bei der Auswahl der Muster für diese Arbeit vermutet, stellt Mikrokernel mit einer Bewertung von **0,72** den Spitzenreiter unter den universell einsetzbaren Strukturmustern dar. Durch seine exzellente Modularität bei trotzdem vergleichsweise geringer Komplexität, sowie guten Bewertungen in vielen anderen Kriterien ist es in den Augen des Autors der klare Sieger dieses Vergleichs. Die Stärken liegen vor allem in der Portabilität und der Erweiterbarkeit.

#### Layers

Das etwas allgemeiner definierte Muster Layers schneidet in vielen Qualitäten nicht viel schlechter ab als Mikrokernel. Für die Evolvability ergibt sich eine Bewertung von **0,61**.

#### Pipes and Filters

Die beste Bewertung aller Qualitäten überhaupt erreicht Pipes and Filters mit Variability. Durch die dynamische Anordnung der Filter ist es für diverse unterschiedliche Aufgaben geeignet, ohne geändert werden zu müssen. Auch die anderen Qualitäten haben ausnahmslos gute Bewertungen. Für die Evolvability wurde eine Bewertung von **0,59** errechnet.

#### Inversion of Control

Auch Inversion of Control, realisiert durch Frameworks, erringt mit einer Bewertung von **0,57** einen der vorderen Plätze. Es schlägt Mikrokernel in Verständlichkeit und Variabilität und ist auch bzgl. der anderen Qualitäten überdurchschnittlich bewertet.

### Client-Server

Mit einer Bewertung von **0,51** befindet sich Client-Server im vorderen Mittelfeld. Seine Stärken liegen in der Erweiterbarkeit und Testbarkeit, was auf die klare Schnittstelle zurückzuführen ist.

### Plug-In

Ein weiteres Muster im vorderen Mittelfeld ist Plug-In mit einer Bewertung von **0,45**. Erweiterbarkeit und Variability sind seine Paradedisziplinen.

### Broker

Mit einer Evolvability-Bewertung von **0,39** führt Broker das Mittelfeld an. Es kann mit Vorteilen in Analysierbarkeit und Änderbarkeit aufwarten, die restlichen Bewertungen liegen im Mittelfeld.

### Proxies

Mit einer durchweg mittelmäßigen Bewertung wartet das Muster Proxies auf. Lediglich in der Analysierbarkeit wird mit 0,5 ein vorderer Platz belegt. Für die Evolvability ergibt sich eine Bewertung von **0,37**.

### Presentation-Abstraction-Control

Dieses Muster ordnet sich mit einer Bewertung von **0,35** bzgl. Evolvability ebenfalls im Mittelfeld ein. Auch die anderen Qualitäten sind mittelmäßig bewertet.

### Model-View-Controller

Die Stärken von Model-View-Controller liegen in der Erweiterbarkeit, Portabilität, Testbarkeit und Variability, allerdings sind die Auswirkungen insgesamt relativ gering, so dass für die Evolvability lediglich ein Wert von **0,31** erreicht wird.

### Blackboard

Die Stärke des Blackboards liegt aufgrund der Zentralisierung in der Testbarkeit, auf die restlichen Qualitäten hat dieses Muster nur geringe Einflüsse. Die Bewertung der Evolvability liegt bei **0,29**.

### Dependency Injection

Mit einer Bewertung von **0,24** in der Evolvability liegt die Stärke in der Änderbarkeit, die restlichen bewerteten Qualitäten werden nur gering bis mittelmäßig unterstützt. Eine Schwäche ist mit einer Bewertung von -0,17 die Testbarkeit.

### Fassade

In den Qualitäten Testbarkeit und Verständlichkeit erringt Fassade gute und sehr gute Bewertungen, die restlichen Qualitäten werden nicht besonders beeinflusst. Somit ergibt sich für die Evolvability eine Bewertung von **0,23**.

### Reflection

Die sehr guten Bewertungen für Änderbarkeit und Variability werden durch negative Bewertungen für Portability, Testbarkeit und Verständlichkeit erkauft. Reflection sollte daher nur an neuralgischen Punkten eingesetzt werden. Für die Evolvability ergibt sich eine Bewertung von **0,19**.

### Implicit Invocation

Die größten Vorteile bietet Implicit Invocation mit einer Bewertung von 0,5 bzw. 0,44 in Testbarkeit und Änderbarkeit, die anderen Qualitäten werden nur unwesentlich beeinflusst. Für die Evolvability ergibt sich daraus eine Bewertung von **0,15**.

### Repository

Repository ist wohl das Muster mir dem geringsten Einfluss auf die gewählten Qualitäten. Mit 0,34 bzw. 0,33 hat es keine besonders großen Auswirkungen auf Portability und Testbarkeit, obwohl das die beiden am stärksten beeinflussten Qualitäten sind. Dennoch hat eine zentrale Persistenzkomponente nicht von der Hand zu weisende Vorteile, falls einmal Änderungen an den Speicherstrukturen notwendig werden. Für die Evolvability ergibt sich eine Bewertung von **0,12**.

#### Adapter

Der Adapter ist speziell dazu da, um vorhandene Komponenten wiederverwendbar zu machen, indem er ansonsten inkompatible Schnittstellen nutzbar macht. Das spiegelt sich eindeutig in der Bewertung wieder: Einer guten Bewertung der Wiederverwendbarkeit stehen geringe Einflüsse auf andere Qualitäten gegenüber, lediglich die Testbarkeit hat einen negativen Einfluss. Für die Evolvability ergibt sich daraus eine Bewertung von **0,12**.

#### Nebenläufigkeit

Mit einer Bewertung von **0,06** hat Nebenläufigkeit in diesem Vergleich die geringsten Auswirkungen auf die Evolvability. Unter den anderen Qualitäten fällt lediglich die negative Bewertung der Testbarkeit ins Gewicht. Allerdings ist die Kernqualität dieses Musters, die Ausführungsgeschwindigkeit auf modernen Systemen mit mehreren Prozessorkernen, hier auch nicht bewertet worden, da sie keinen Einfluss auf die Evolvability hat.

### 5.1.3 Nutzen der Bewertungen

Die hier erarbeitete Bewertung soll von Softwarearchitekten als Hilfe bei der Auswahl der in einem Projekt einzusetzenden Muster benutzt werden. Hierbei darf der Softwarearchitekt allerdings die Auswahl nicht allein auf diese Bewertung gründen, er muss diese immer im Kontext seines Projektes sehen. Die Bewertungen spiegeln die Auswirkungen auf den Einflussbereich des jeweiligen Musters wieder. Die Bewertung eines Muster mit globalen Auswirkungen auf die Software, wie beispielsweise Layers, ist also wesentlich relevanter für die daraus resultierende Gesamtsoftware als die Bewertung eines Musters mit nur sehr lokalen Auswirkungen, wie beispielsweise eines lokal eingesetzten Adapters. Kommt der Adapter allerdings an einer Schlüsselposition in der Architektur der Gesamtsoftware zum Einsatz, so wird auch die Bewertung des Adapters wesentlich relevanter für die Software im Ganzen.

Bei einer Kombination mehrer Muster in einem Entwurf könnten die Einfüsse auf die Qualitäten von der Summe der Bewertungen der einzelnen eingesetzten Muster in nicht vorhersehbarem Maße abweichen, da auch das Zusammenspiel der Muster untereinander Auswirkungen auf die Kriterien der Hierarchie haben kann.

## *5.2 Klassifikation*

Das Ziel eines Softwareentwurfs sind immer Qualitäten, die eine Software aufweisen soll. Da die gewünschten Qualitäten oft schwer fassbar sind, werden beim Entwurf meist Prinzipien umgesetzt, die die gewünschten Qualitäten unterstützen. Bei der Umsetzung der Prinzipien wird auf Architekturstile und -muster zurückgegriffen. Letztendlich ist also das Interessante der Einfluss der Muster auf die Qualitäten, nicht auf die Prinzipien. Aus diesem Grund wird sich die Klassifikation entgegen der Aufgabenstellung nur auf die Bewertung der Qualitäten stützen.

### 5.2.1 Entstehung der Klassen

Aus den Ergebnissen der Bewertung ergeben sich insgesamt 4 Klassen. Die erste Klasse "Allrounder" soll aus denjenigen Mustern bestehen, die großteils gute Bewertungen erreicht haben und somit als grundlegende Muster für einen Entwurf dienen können. Diese sind **Client-Server**, **Layers**, **Batchprogramme**, **Pipes and Filters**, **Inversion of Control** und **Mikrokernel** (siehe auch Abbildung 26: Darstellung der Klasse Allrounder im Anhang). Wenn man pro Muster bis zu 2 Qualitäten mit geringerem Einfluss zulässt, so zeichnet sich bei einer Bewertung von 0,45 eine untere Grenze für diese Klasse ab.

Desweiteren erscheint es sinnvoll, eine "Spezialisten"-Klasse mit Mustern zu bilden, die sich auf eine oder zwei Qualitäten besonders positiv auswirken und die anderen nur unwesentlich beeinflussen. Dadurch eignen sich diese Muster speziell für den Einsatz an kritischen Stellen im Entwurf. Zu dieser Klasse gehören **Nebenläufigkeit**, **Fassade**, **Adapter**, **Dependency Injection** und **Reflection** (siehe auch Abbildung 27: Darstellung der Klasse Spezialisten im Anhang). Die Qualität, auf die sich die Nebenläufigkeit besonders positiv auswirkt, ist die Laufzeiteffizienz auf modernen Systemen mit mehreren Prozessorkernen, die in dieser Arbeit nicht bewertet wurde. Abgesehen von der Nebenläufigkeit zeichnen sich die Muster dieser Klasse dadurch aus, dass sie in mindestens einer Qualität eine Bewertung von mehr als 0,6 und in maximal 2 Qualitäten über 0,45 erhalten.

Eine dritte Klasse "Low Impact" wird aus 2 Mustern gebildet, die in keiner der hier behandelten Qualitäten eine Bewertung von mehr als 0,5 erhalten hat. Diese sind **Repository** und **Implicit Invocation** (siehe auch Abbildung 28: Darstellung der Klasse Low Impact im Anhang). Obwohl beide durchaus einen wichtigen Zweck erfüllen, bleibt ihr qualitativer Einfluss gering.

Die bisher nicht klassifizierten Muster **Blackboard**, **Model-View-Controller**, **Presentation-Abstraction-Control**, **Broker**, **Proxies** und **Plug-In** wirken sich jeweils auf mehrere Qualitäten in größerem Umfang positiv aus. Sie werden in der vierten Klasse "Sonstige Muster" zusammengefasst (siehe auch Abbildung 29: Darstellung der Klasse Sonstige Muster im Anhang).

# 6 Fazit und Ausblick

Aufgabe dieser Arbeit war die Bewertung und Klassifikation von Architekturstilen und -mustern nach qualitativen Gesichtspunkten, wobei die ausgewählten Qualitäten Subcharakteristiken von Evolvability sein sollten. Dazu wurde zuerst eine repräsentative Auswahl von Mustern getroffen, die für die Evolvability besonders interessant erschienen oder sehr bekannt sind.

Um eine Bewertung dieser Muster durchführen zu können, wurde eine Hierarchie von Kriterien erstellt. Diese wurde so angelegt, dass die Auswirkungen der Muster auf die grundlegenden Kriterien einfach zu beurteilen sind. Durch eine Gewichtung aller Abhängigkeiten in der Hierarchie kann nun der Einfluss der Muster auf die Qualitäten höherer Hierarchieebenen berechnet werden. So kann eine fundierte Aussage über den Einfluss auf die ansonsten schwer fassbare Evolvability gemacht werden. Durch eine weitergehende Evaluation der Gewichte kann die Relevanz des Ergebnisses gesteigert werden.

Diese Art der Bewertung ist aber nicht nur für Muster, sondern für alles einsetzbar, was sich auf die Qualitäten auswirkt. Da der Autor dies frühzeitig erkannte, hat er die in der Hierarchie verwendeten Prinzipien, Attribute und Qualitäten nicht auf solche beschränkt, die durch Muster beeinflusst werden. Somit ist die in dieser Arbeit vorgestellte Hierarchie unverändert für alles einsetzbar, was sich auf diese Kriterien oder einen Teil von ihnen auswirkt. Ein weiterer Vorteil ist, dass auch der Umfang des Einflusses durch die Beträge der Bewertungen sichtbar wird. Sind diese nahe Null, so ist der Einfluss gering. Eine Aufstellung vergleichbarer Hierarchien für weitere Topziele und ggf. eine Verknüpfung dieser zu einer allumfassenden Hierarchie kann Inhalt einer weiterführenden Arbeit sein. Auch eine Bewertung weiterer Architekturlösungsansätze ist neben der Verbesserung und Erweiterung der Hierarchie eine sinnvolle Aufgabe für die Zukunft.

Bei der Betrachtung der Ergebnisse der Bewertung der Muster fiel schnell auf, dass vor allem die Strukturmuster einen breiten Einfluss auf viele Qualitäten haben, während andere Muster sich oft nur punktuell auf einzelne oder wenige Qualitäten auswirken. Dementsprechend wurden die Klassen definiert.

Mit dem Ergebnis dieser Arbeit steht Softwarearchitekten nun eine sehr hilfreiche Grundlage für die Auswahl der richtigen Muster beim Entwurf einer neuen Software zur Verfügung. Dennoch kann es nicht das einzige Kriterium sein, da auch das Zusammenspiel der einzelnen Muster sowie die Anwendbarkeit bei gegebenen Rahmenbedingungen von essenzieller Bedeutung sind.

# 7 Anhang

## *7.1 Ergebnisse der Bewertung*

| Muster \ Kriterien | Change Impact | Kapselung | Komplexität | Konsistenz | Kopplung | Korrektheit | Modularität | Patency | Separation of Concerns | Typifikation | Vollständigkeit | Analysierbarkeit | Änderbarkeit | Einhaltung von Standards | Erweiterbarkeit | konzeptionelle Integrität | Portability | Testbarkeit | Traceability | Variability | Verständlichkeit | Wiederverwendbarkeit | Evolvability |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Client-Server | 0 | 0 | -3 | n.b. | 0 | n.b. | 3 | n.b. | 1 | n.b. | n.b. | 0,25 | 0,48 | n.b. | 1,14 | n.b. | 0,83 | 1,17 | n.b. | 0,75 | 0,67 | 0,38 | 0,51 |
| Layers | 0 | 0 | -3 | n.b. | 0 | n.b. | 3 | n.b. | 2 | n.b. | n.b. | 0,25 | 0,5 | n.b. | 0,85 | n.b. | 1,83 | 1,33 | n.b. | 0,75 | 0,73 | 0,38 | 0,61 |
| Repository | 0 | 0 | 1 | n.b. | 0 | n.b. | 1 | n.b. | 2 | n.b. | n.b. | 0,08 | 0,03 | n.b. | 0,17 | n.b. | 0,34 | 0,33 | n.b. | 0,25 | 0,07 | 0,13 | 0,12 |
| Blackboard | 0 | 0 | -2 | n.b. | -1 | n.b. | 1 | n.b. | 2 | n.b. | n.b. | 0,17 | 0,34 | n.b. | 0,42 | n.b. | 0,54 | 1 | n.b. | 0,25 | 0,53 | 0,13 | 0,29 |
| Batchprogramme | 0 | 3 | -3 | n.b. | -3 | n.b. | 3 | n.b. | 0 | n.b. | n.b. | 1,08 | 1,7 | n.b. | 1,6 | n.b. | 0,97 | 1 | n.b. | 0,75 | 1,4 | 0,75 | 0,84 |
| Pipes and Filters | 0 | 2 | -2 | n.b. | 0 | n.b. | 3 | n.b. | 1 | n.b. | n.b. | 0,42 | 0,57 | n.b. | 0,91 | n.b. | 0,89 | 0,67 | n.b. | 2,25 | 0,67 | 0,63 | 0,59 |
| Model-View-Controller | 0 | 0 | 2 | n.b. | 0 | n.b. | 3 | n.b. | 3 | n.b. | n.b. | 0,25 | 0,13 | n.b. | 0,54 | n.b. | 0,74 | 0,67 | n.b. | 0,75 | 0,13 | 0,38 | 0,31 |
| Presentation-Abstraction-Control | 0 | 0 | 3 | n.b. | -2 | n.b. | 3 | n.b. | 3 | n.b. | n.b. | 0,42 | 0,21 | n.b. | 0,61 | n.b. | 0,7 | 0,83 | n.b. | 0,75 | 0,13 | 0,38 | 0,35 |
| Implicit Invocation | -1 | 0 | 0 | n.b. | -3 | n.b. | 0 | n.b. | 0 | n.b. | n.b. | 0,25 | 0,44 | n.b. | 0,25 | n.b. | 0,03 | 0,5 | n.b. | 0 | 0,2 | 0 | 0,15 |
| Nebenläufigkeit | 0 | 2 | 0 | n.b. | 0 | n.b. | 0 | n.b. | 0 | n.b. | n.b. | 0,17 | 0,16 | n.b. | 0,13 | n.b. | 0,13 | -0,5 | n.b. | 0 | 0,13 | 0,25 | 0,06 |
| Inversion of Control | 0 | 2 | -2 | n.b. | -2 | n.b. | 2 | n.b. | 2 | n.b. | n.b. | 0,5 | 0,66 | n.b. | 1,13 | n.b. | 0,85 | 1 | n.b. | 1 | 0,8 | 0,5 | 0,57 |
| Fassade | 0 | 0 | -2 | n.b. | -2 | n.b. | 0 | n.b. | 0 | n.b. | n.b. | 0,17 | 0,43 | n.b. | 0,32 | n.b. | 0,18 | 0,67 | n.b. | 0 | 1 | 0 | 0,23 |
| Adapter | 0 | 2 | 1 | n.b. | -2 | n.b. | 0 | n.b. | 0 | n.b. | n.b. | 0,33 | 0,24 | n.b. | 0,2 | n.b. | 0,08 | -0,17 | n.b. | 0 | 0,13 | 0,63 | 0,12 |
| Dependency Injection | -2 | 3 | 0 | n.b. | -2 | n.b. | 0 | n.b. | 0 | n.b. | n.b. | 0,42 | 0,8 | n.b. | 0,43 | n.b. | 0,21 | -0,17 | n.b. | 0 | 0,33 | 0,38 | 0,24 |
| Broker | -1 | 2 | 2 | n.b. | -3 | n.b. | 2 | n.b. | 1 | n.b. | n.b. | 0,58 | 0,82 | n.b. | 0,69 | n.b. | 0,49 | 0,33 | n.b. | 0,5 | 0,27 | 0,5 | 0,39 |
| Proxies | 0 | 2 | 1 | n.b. | -2 | n.b. | 2 | n.b. | 2 | n.b. | n.b. | 0,5 | 0,43 | n.b. | 0,64 | n.b. | 0,66 | 0,5 | n.b. | 0,5 | 0,4 | 0,5 | 0,37 |
| Mikrokernel | 0 | 2 | -1 | n.b. | -2 | n.b. | 3 | n.b. | 2 | n.b. | n.b. | 0,58 | 0,66 | n.b. | 1,43 | n.b. | 1,85 | 1 | n.b. | 0,75 | 0,73 | 0,63 | 0,72 |
| Reflection | -3 | 0 | 2 | n.b. | 0 | n.b. | 1 | n.b. | 2 | n.b. | n.b. | 0,08 | 1,07 | n.b. | 0,21 | n.b. | -0,19 | -0,17 | n.b. | 1,25 | -0,47 | 0,13 | 0,19 |
| Plug-In | 0 | 2 | 0 | n.b. | 0 | n.b. | 3 | n.b. | 0 | n.b. | n.b. | 0,42 | 0,4 | n.b. | 1,23 | n.b. | 0,65 | 0,17 | n.b. | 1,25 | 0,33 | 0,5 | 0,45 |

## 7.2 Darstellung der Klassen

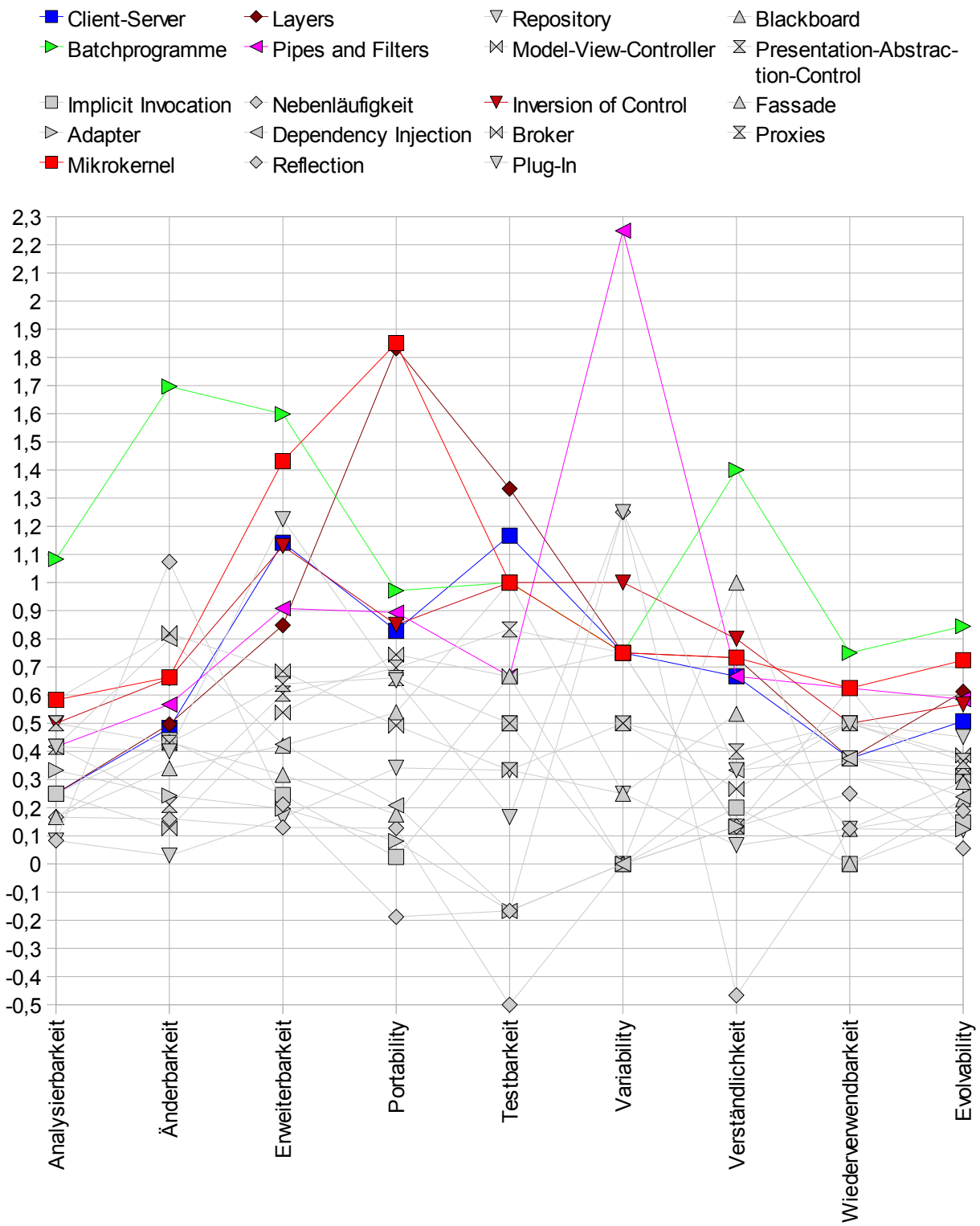


*Abbildung 26: Darstellung der Klasse Allrounder*

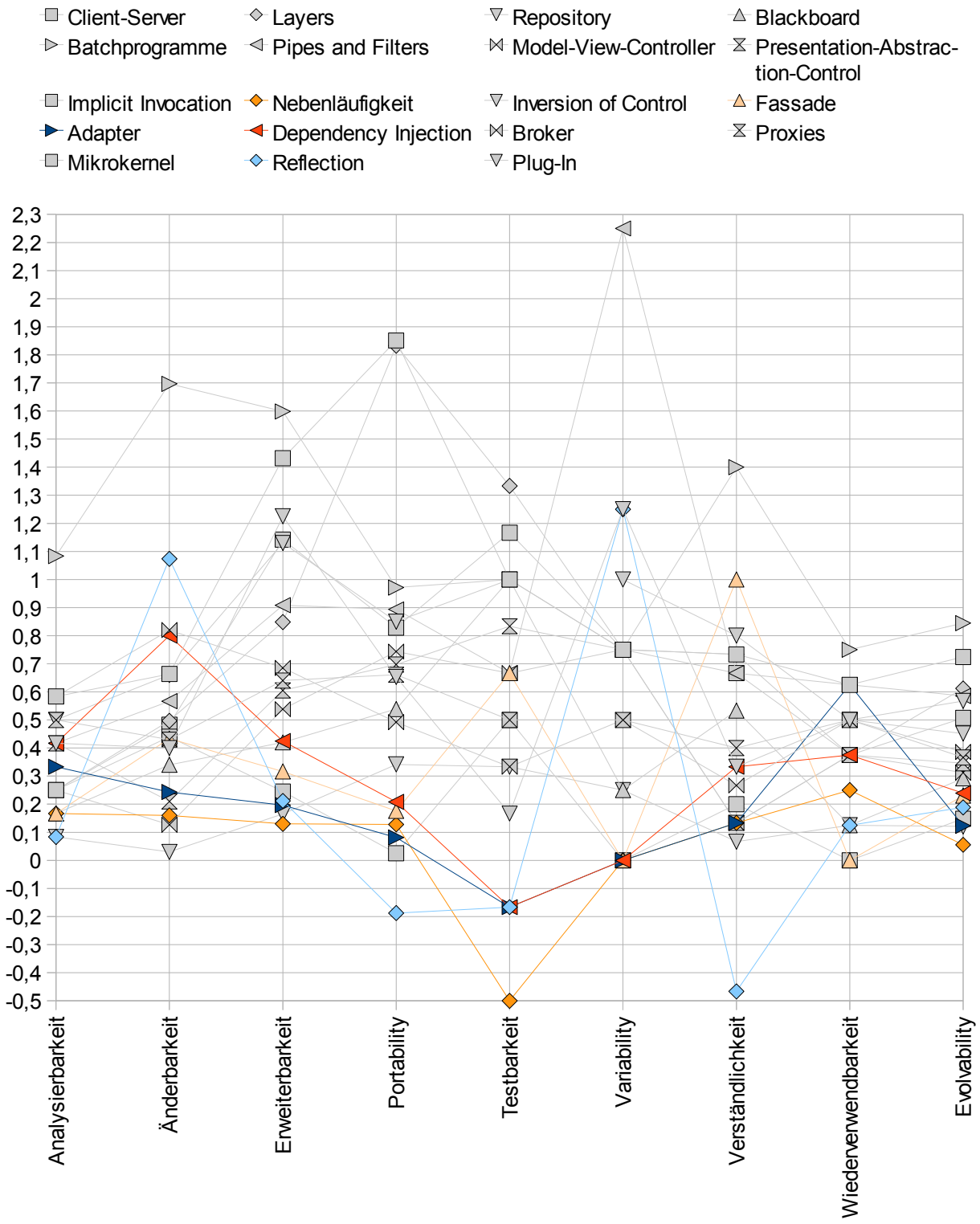


*Abbildung 27: Darstellung der Klasse Spezialisten*

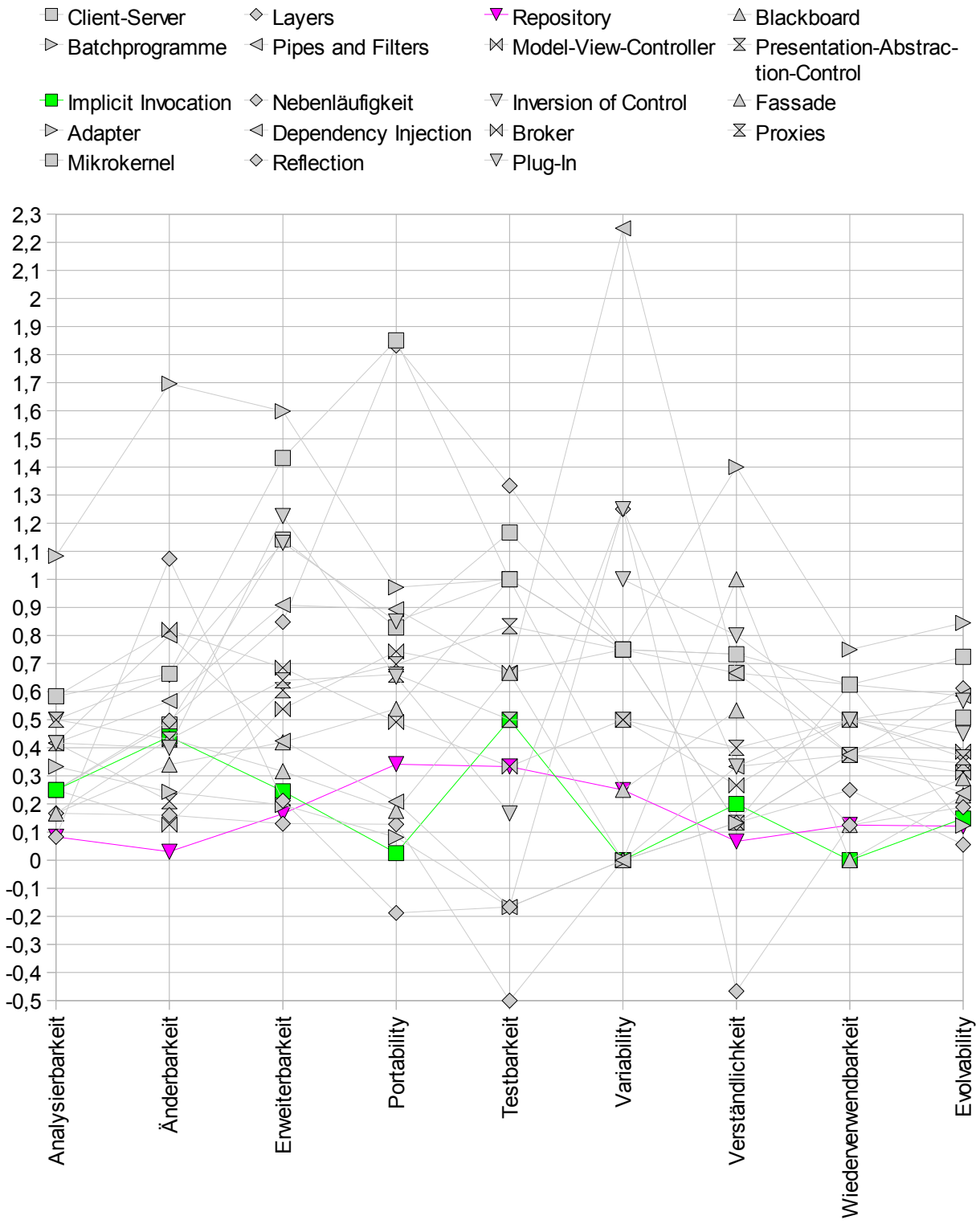


*Abbildung 28: Darstellung der Klasse Low Impact*

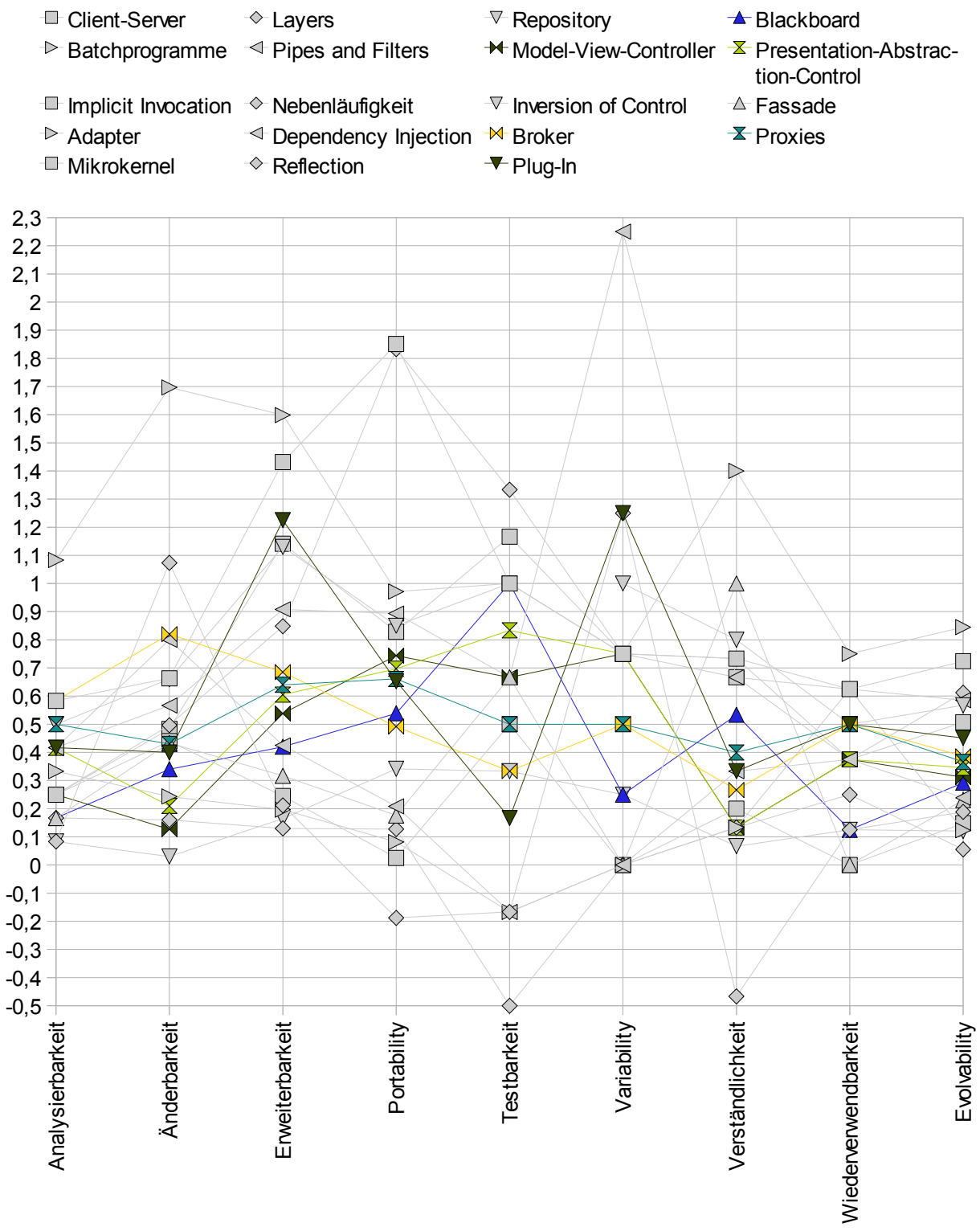


*Abbildung 29: Darstellung der Klasse Sonstige Muster*

# 8 Literaturverzeichnis

[BBR09] Brcina, Robert ; Bode, Stephan ; Riebisch, Matthias: Optimization Process for Maintaining Evolvability during Software Evolution. In: Proc. 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2009), San Francisco, CA, USA, April 13-16, pp. 196-205, IEEE Computer Society, 2009

[BCE07] Breivold, Hongyu Pei ; Crnkovic, Ivica ; Eriksson, Peter J.: Evaluating Software Evolvability. In: Proceedings of the 7th Conferenceon Software Engineering Research and Practice in Sweden (SERPS'07), 2007

[BCK98] Bass, Len ; Clements, Paul ; Kazman, Rick: Software Architecture in Practice. Addison-Wesley, 1998. - ISBN 0201199300

[BME07] Booch, Grady ; Maksimchuk, Robert A. ; Enge, Michael W.: Object-Oriented Analysis and Design with Applications. Addison-Wesley, 2007. - ISBN 9780201895513

[BMRSS96] Buschmann, Frank ; Meunier, Regine ; Rohnert, Hans ; Sommerlad, Peter ; Stal, Michael: Pattern-Oriented Software Architecture : A System of Patterns. John Wiley & Sons, 1996. - ISBN 0471958697

[FowDI] Fowler, Martin: Inversion of Control Containers and the Dependency Injection pattern. http://martinfowler.com/articles/injection.html, abgerufen am 15.02.2010

[FowIoC] Fowler, Martin: Inversion of Control. http://martinfowler.com/bliki/InversionOfControl.html, abgerufen am 15.02.2010

[GF94] Gotel, Orlena C. Z. ; Finkelstein, Anthony C. W.: An analysis of the requirements traceability problem. In: First International Conference on Requirements Engineering (ICRE'94), Colorado Springs, S. 94-101, IEEE Computer Society Press, 1994

[GHJV95] Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: Design Patterns : Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995. - ISBN 0201633612

[GP70] Gauthier, Richard L. ; Ponto, Stephen D.: Designing systems programs. Prentice-Hall, 1970. - ISBN 0132019620

[IEEE 610.12] IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990.

[ISO9126] ISO/IEC 9126-1 Software Engineering - Product Quality -Part 1: Quality

Model, June 2001.

[msdn998478] Microsoft Corporation: Deployment Patterns. http://msdn.microsoft.com/en-us/library/ms998478.aspx, abgerufen am 15.02.2010

[MVN06] Manolescu, Dragos ; Voelter, Markus ; Noble, James: Pattern Languages of Program Design 5. Addison Wesley, 2006. - ISBN 0321321944

[SB03] Simão, Régis P.S. ; Belchior, Arnaldo D.: Quality Characteristics for SoftwareComponents : Hierarchy and Quality Guides. In: Component-Based Software Quality, pages 184–206, 2003

[SG96] Shaw, Mary ; Garlan, David: Software architecture : perspectives on an emerging discipline. Prentice-Hall, 1996. - ISBN 0131829572

[wikiCS] Diverse: Client-Server-Modell. http://de.wikipedia.org/wiki/Client-Server, abgerufen am 15.02.2010

[wikiP2P] Diverse: Peer-to-Peer. http://de.wikipedia.org/wiki/Peer-2-Peer, abgerufen am 15.02.2010

# Abbildungsverzeichnis

Abbildung 1: Client-Server....4
Abbildung 2: Layers....4
Abbildung 3: Tiers....4
Abbildung 4: Repository....5
Abbildung 5: Blackboard....5
Abbildung 6: Batchprogramm....6
Abbildung 7: Pipes and Filters....7
Abbildung 8: Model-View-Controller....8
Abbildung 9: Presentation-Abstraction-Controller....9
Abbildung 10: Inversion of Control....10
Abbildung 11: Mikrokernel....12
Abbildung 12: Hierarchie der Bewertungskriterien....16
Abbildung 13: Beispiel für Client-Server....24
Abbildung 14: Beispiel für Layers....26
Abbildung 15: Beispiel für Repository....27
Abbildung 16: Beispiel für Blackboard....28
Abbildung 17: Beispiel für Batchprogramm....30
Abbildung 18: Beispiel für Pipes and Filters....31
Abbildung 19: Beispiel für Model-View-Controller....33
Abbildung 20: Beispiel für Mikrokernel....42
Abbildung 21: Dialog zum Beispiel für Reflection....44
Abbildung 22: Klassendiagramm zum Beispiel für Reflection....44
Abbildung 23: Zuordnung der Ebenen der Hierarchie auf die Tabellenblätter....48
Abbildung 24: Bewertung der Evolvability....50
Abbildung 25: Ergebnisse der Bewertung....51
Abbildung 26: Darstellung der Klasse Allrounder....59
Abbildung 27: Darstellung der Klasse Spezialisten....60
Abbildung 28: Darstellung der Klasse Low Impact....61
Abbildung 29: Darstellung der Klasse Sonstige Muster....62

# Tabellenverzeichnis

Tabelle 1: Bewertung von Client-Server....25
Tabelle 2: Bewertung von Layers, Tiers....27
Tabelle 3: Bewertung von Repository....28
Tabelle 4: Bewertung von Blackboard....29
Tabelle 5: Bewertung von Batchprogramme....31
Tabelle 6: Bewertung von Pipes and Filters....32
Tabelle 7: Bewertung von Model-View-Controller....33
Tabelle 8: Bewertung von Presentation-Abstraction-Control....34
Tabelle 9: Bewertung von Event Based, Implicit Invocation....35
Tabelle 10: Bewertung von Nebenläufigkeit....36
Tabelle 11: Bewertung von Inversion of Control....37
Tabelle 12: Bewertung von Fassade....38
Tabelle 13: Bewertung von Adapter....39
Tabelle 14: Bewertung von Dependency Injection....40
Tabelle 15: Bewertung von Broker....41
Tabelle 16: Bewertung von Proxies....42
Tabelle 17: Bewertung von Mikrokernel....43
Tabelle 18: Bewertung von Reflection....45
Tabelle 19: Bewertung von Plug-In....46
Tabelle 20: Ergebnisse der Bewertung....50

## *8.1 Selbständigkeitserklärung*

Hiermit erkläre ich, dass ich die Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderweitigen fremden Äußerungen entnommen wurden, sind als solche kenntlich gemacht. Ferner erkläre ich, dass die Arbeit noch nicht anderweitig als Prüfungsleistung verwendet wurde.

Ilmenau, den 22.02.2010

_____

Ralf Stollberg